ABSTRACT

YU, XIANQING. Enhancing Resilience to Compromise in a Public Cloud. (Under the direction of Mladen A. Vouk.)

Cloud computing is becoming increasingly more popular as it provides reliability, flexibility, scalability, elasticity and cost saving to various users. However, cloud cyber-attacks challenge security and privacy of users in a cloud. In this dissertation, we explore different dimensions of possible threats to systems in a public cloud and develop three novel models to enhance the attack resilience in a public cloud. We find that isolation, access control, and efficient encryption can solve many of the issues.

We first investigated security of storage operated in a cloud. Specifically, in our case-study, we study a distributed file system and a distributed parallel data processing application, called Hadoop. It is widely used in both research and commercial services and is available in many clouds. However, Hadoop was originally designed to run in a private environment. "Out-of-the-box" Hadoop shows that in a shared environment its overloaded authentication keys and lack of fine-grained access control are vulnerabilities that need addressing. We developed and prototyped SEHadoop model to fix the vulnerabilities and enhance the security of Hadoop in a cloud environment by improving isolation level and enforcing least access control among Hadoop components. Improved isolation and access control are in general the key to solution of many cloud-based security issues that may arise when porting into a cloud applications not originally designed for that environment.

We then investigate how deduplication technology can operate on encrypted data stored in a cloud-based storage system (CSS). Deduplication is used to improve storage utilization on CSS. Applying deduplication to encrypted CSS files has challenges as CSS is usually shared among many users. Brute-force dictionary attack and side-channel attack are possible. We designed and implemented a model we call SafeDedup for secure and flexible way of deduplicating encrypted data. This model can also be used in the context of fine granularity file sharing in a CSS environment.

The third part of this work is a discussion of the security of a multi-cloud system. It may be cost-efficient and practical to integrate resources of multiple clouds. However, different clouds are usually managed by different organizations with different security policies and management platforms. When some components of a multi-cloud system are compromised, attackers can potentially have a high privilege that impacts the rest of system. We developed a model we call SECross for fine-grain database access policy of SECross components, an access rule among SECross components, a method for users to access computing machines and security policy on each SECross management node. We discussed how SECross may be able to resist potential

attacks when a SECross component is compromised.

Enhancing Resilience to Compromise in a Public Cloud

by
Xianqing Yu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2016

APPROVED BY:

Guoliang Jin	Vincent W. Freeh
Rada Y. Chirkova	Billy Williams

Mladen A. Vouk
Chair of Advisory Committee

DEDICATION

To my parents.

BIOGRAPHY

Xianqing was born and raised in China. He attended Hefei No. 1 high school from 2001 to 2004. He started his college in Shanghai University in 2004 and completed his Bachelor of Engineering degree in automation in 2008. After that, he began his graduate studies at North Carolina State University in Raleigh, NC, the USA in 2008 and had his Master degree in Computer Networking in 2010. He started to pursue a Computer Science Ph.D. degree at North Carolina State University in 2010. And he worked with Dr. Peng Ning and Dr. Mladen A. Vouk for his research in the Computer Science department. His research focused on cloud computing security.

ACKNOWLEDGEMENTS

This work is supported in part through by NSF grant 0910767, 1330553 and 1318564, the U.S. Army Research Office (ARO) grant W911NF-08-1-0105, the NSA Science of Security Lablet, and by the IBM Shared University Research, and IBM Fellowships program funding.

I am greatly thankful to my advisors, Dr. Mladen A. Vouk, and Dr. Peng Ning, for the tremendous guidance in my Ph.D. study. Thank you for the inspiring and helpful discussions in my research. I also would like to thank my committee, Dr. Guoliang Jin, Dr. Vincent W. Freeh, Dr. Rada Y. Chirkova and Dr. Billy Williams, for their time and effort serving. Thank you for the insightful comments and valuable feedback in my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Cloud computing is a technology with promising scalability, Quality of Service (QoS), reliability and cost efficiency. It allows users to share computing hardware and software at a very large scale. However, resources sharing brings with it a number of potential threats, especially in a public cloud [1–6]. In this dissertation, we examine three types of issues that may arise in a publicly shared space - use of software that may not have been designed for cloud use, use of public storage systems when we desire to use encrypted data, and some of the issues that may arise in multi-cloud systems operated by different vendors. In the first case, we studied the use of Hadoop in a cloud, a popular environment for data analytics that originally was not designed for shared environment. In the second case we investigated the use of deduplication on encrypted data. In the third case, we examined and discussed impact that distribution of an application or workflow over several public clouds may have on security.

1.1 Hadoop in a Public Cloud

Map-Reduce is often used in the context of big data. Hadoop is an open-source implementation of the Google Map-Reduce framework [7]. Many use Hadoop to analyze very large data sets. A large Hadoop cluster can have more than 42,000 computing nodes [8, 9]. There have been a number of successful efforts to move Hadoop into cloud [10–14]. For example, Amazon Elastic MapReduce (Amazon EMR), Cloud::hadoop [11] and Joyent solution for Hadoop [10] are commercial variants.

However, Hadoop was originally designed to run in a well controlled private environment. When Hadoop operates in a public cloud, there are challenges to original Hadoop security mechanisms. In a concurrent resource sharing cloud, hardware resources (e.g. CPU, memory, hard disks, network card) and software resources, (e.g. hypervisor, operating system) can be shared among cloud users [4]. In such a resource sharing environment, security often relies on the high-level of isolation of users' workloads [15, 16]. This isolation is often enforced through

virtualization technology. Typically a hypervisor (and underlying O/S if present) is the highest privilege software to manage all hardware resources, and to enforce the isolation. Unfortunately, a hypervisor itself can be a target of attacks [17–21]. In his work, F. Rocha [22] demonstrated how a compromised hypervisor can obtain passwords, cryptographic keys, files and other confidential data. In [23] S. Bugiel et al. discussed various unsafe publicly available and widely used Amazon Machine Images (AMIs). T. Ristenpart et al. showed how to determine if two instances were co-resident on the same physical machine and thus able to launch side-channel attack in Amazon EC2 [24]. These and similar internal cloud attacks can bypass out-of-the-box Hadoop security mechanisms raising concerns that Hadoop in its present form may not be able to maintain the same security level in a public cloud as it does in a protected environment. We developed and implemented a Hadoop model, we call SEHadoop model, intended to improve the resilience of Hadoop to compromises, that may occur in share environments, in two major aspects: enhancing isolation level among Hadoop components and enforcing the least privilege access control on Hadoop processes. High isolation level among Hadoop components can prevent compromised Hadoop processes from compromising the rest of Hadoop. Least privilege access control can limit the range of data a compromised Hadoop process can access. We have following contributions:

- We explored potential vulnerabilities when "out-of-the-box" Hadoop runs in a public cloud.
- SEHadoop model is designed and implemented to improve the compromise resilience of Hadoop. We redesigned Block Token and Delegation Token and developed SEHadoop runtime model to improve the isolation level among Hadoop components and enforce the least access control policy.
- We experimentally accessed how SEHadoop resists of compromises. The overhead of SEHadoop is evaluated.

1.2 Encryption and Deduplication in Cloud Storage Service

Cloud-based storage Software-as-a-Service (SaaS), such as Dropbox, Google Drive and Mozy, provides convenient and cost-saving storage service to various customers. Use of deduplication to increase utilization of physical storage resources is common. However, deduplication of encrypted data may be a challenge. Deduplication ensures that only one single copy of a file is saved in CSS space. This saves space, and may help protect integrity of the data since there are fewer physical places that may need protection. On the other hand, users may prefer to encrypt their data just in case.

Unfortunately, performing deduplication can leak data sharing information as well as the data itself. On the other hand, use of a unique encryption key for every user would dramatically undermine the effectiveness of deduplication. Sharing a key among users can improve the result of deduplication, but the system may now become vulnerable to malicious insiders. Many traditional encryption methods do not work well with deduplication [25–27].

Adding to the problem are challenges in data sharing of encrypted files in a cloud [28–31]. Data sharing parties need keys to access shared encrypted data. However, data sharing privileges are usually dynamically changing. Simply sharing a key fails to achieve resilience and at the same time provide flexibility of access control that may be needed (e.g. changing read and write privilege to read only). A secure, flexible and light-weight data sharing mechanism with fine-grained access control is needed for both sharing encrypted data and for deduplication to be effective.

One approach that tried to solve encryption-deduplication problem is convergent encryption (CE), introduced by Douceur et al [32]. In CE, a key is derived from file content, for example, by using a cryptographic hash function: Key $\leftarrow$ Hash(file). Then the key can be used to encrypt data using a deterministic symmetric encryption scheme. If two files have the same content, the same encryption key and ciphertexts can be generated, and the same ciphertexts can ensure deduplication. CE and its variants have been deployed in many systems [26, 33–35]. However, CE is vulnerable to brute-force dictionary attacks if the target file content is drawn from a limited space. This is the so-called predictable files problem [25]. An attacker can use the public known CE method to encrypt all possible plaintexts to find whether there is a match with the target ciphertext. To overcome this type of the attack, the content of a file has to be unpredictable. This may be hard to guarantee in the real world. Bellare et al. [25] use a key server to create encryption keys for all users, so attackers can not derive the keys by themselves without compromising the key server. This is an improvement, but it has the similar security weakness when sharing a key in one group that may have a malicious member. Malicious insiders can use the key server to perform online brute-force dictionary attack. Furthermore, users from different key server groups cannot deduplicate their data on the CSS. If the size of a group is small, deduplication effectiveness is limited. If the size is large, it increases the possibility of a malicious attack. We believe there currently are no method that let data owners effectively and securely share encrypted and deduplicated data with others on the CSS.

A major challenge is that data protection and deduplication have conflicting requirements. Data protection prevents attackers from retrieving plaintexts, and data deduplication requires CSS to know which files are shared by users in order to perform deduplication. However, when attackers can obtain file sharing information [27], and data encryption method of the CSS is publicly known, online or offline brute-force dictionary attack can be launched to retrieve

predictable data on the CSS.

As part of current work, we designed and implemented a model we call SafeDedup to protect the confidentiality of data, match space savings original deduplication techniques may provide, and provide data owners with secure and flexible ways to share data on a CSS system. Our model can resist brute-force dictionary attacks and the threats, and deduplication of our model can be applied to all users instead of only within a group. Our model leverages the privileges of a hypervisor to perform sensitive operations and enforce isolation. We have following contributions:

- A novel method to separate data and file sharing information from a CSS system by leveraging the privilege of a hypervisor. It allows the CSS system to work as normal, such as managing users, data, access control., but does not allow the CSS system to access data or file sharing information among users.
- A compromise-resistant encryption method is developed to encrypt data and perform deduplication at the same time.
- A data access protocol is implemented to allow users to access and share encrypted data with others in a fine granularity access control.

1.3 Issues with Multi-cloud

According to the National Institute of Standards and Technology (NIST) definition, "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [36]. There are many popular and widely used commercial cloud platforms. For example, IBM Softlayer [37], Amazon Amazon Elastic Compute Cloud (EC2) [38], Microsoft Cloud [39], Google Cloud Platform [40], and so on. There are far fewer production level solutions developed in academia (e.g. apache virtual computing lab (VCL) [41, 42]. These cloud platforms provide a range of cloud services, from infrastructure as a service (IaaS), to platform as a service (PaaS), to software as a service (SaaS).

Driven by scalability, cost efficiency and reliability concerns, some have developed multi-cloud environments [43–47]. Guarantees that such environments may carry can be extremely important for users who have highly volatile cloud service demands [15]. For instances, schools have special characteristics which can challenge a single cloud solution. On one hand, urging exams or during times when homework or projects are due, there may be extra high demand on cloud-based services. On the other hand, there may be little or no demand during holidays or summer. A single cloud has to maintain enough capacity to satisfy peak requests but this wastes resources

during low workload periods. A multi-cloud platform can act as an overflow buffer, and a diversity engine that may be able to improve not only on scalability but also reliability (e.g. via fail-over mechanisms) and security.

A multi-cloud platform usually shares its hardware resources and software resources (e.g. hypervisor) with more public users than a single cloud system does, thus, its scalability and cost efficiency can be better than a single cloud solution. Typically a multi-cloud system is deployed using several cloud providers. These cloud providers may have their own architecture, management system, security policies, users, etc. Therefore, it may be difficult for a multi-cloud system to ensure security. For example, a multi-cloud system may share a lot of resources [37–39], and it may be hard to control who the system shares resources with. Attackers can leverage this to launch a range of attacks. As discussed in Section 1.1, hypervisor vulnerabilities can be used [17–21] and side-channel cloud attacks can be launched [22–24].

We discussed a model we call SECross to compensate for compromised multi-cloud system components in a public cloud. We accessed our model using VCL and IBM Softlayer cloud. We have following contributions:

- We discussed The advantages and potential vulnerabilities of a multi-cloud system.
- We designed and implemented SECross model to improve the compromise resilience of a multi-cloud system, including fine-grained access control for the database, the approach for user authentication and access rules for SECross components.
- We analyzed the security impacts of SECross while facing various threats.

Chapter 2

SEHadoop: Enhancing Security of Hadoop in a Public Cloud

2.1 Hadoop Background

Map-Reduce is a programming model for processing large data [48]. There are three phases in a Hadoop Map-Reduce workflow cycle: map, combine (optional), and reduce. Architecturally, using Hadoop 2.0.1 as an example, Hadoop consists of four parts: Hadoop Common, which is common utilities to support other modules, Hadoop YARN, which does resource management, Hadoop MapReduce framework, which does data processing, and Hadoop Distributed File System (HDFS), which is a data storage system [49]. HDFS and YARN use Block Token and Delegation Token to authenticate and authorize Hadoop users. In the context of the vulnerabilities of interest in this dissertation, we discuss HDFS, YARN, Block Token and Delegation Token in more details.

2.1.1 HDFS and Block Token

HDFS is a highly fault-tolerant high-throughput distributed file system. It follows master/slave architecture as illustrated in Figure 2.1. The one named Name Node is the master. Data Nodes work as slaves and typically run on different computers. Name Node performs namespace operations, maps a file to a storage block, maps a storage block to a Data Node, enforces access control, generates access tokens, and similar. A Data Node is responsible for providing services, such as read or writes, to HDFS clients.

A Block Token is used by a process to access data on a Data Node. During HDFS initialization, Name Node randomly generates a symmetric key and distributes it to Data Nodes when Data Nodes are booting up and contacting the Name Node for registration for the first time. The key

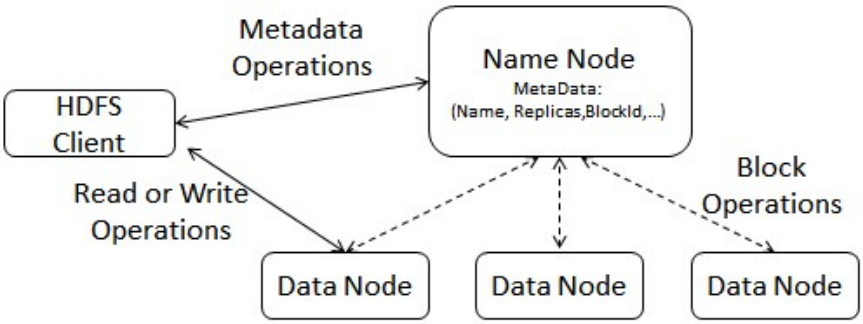

Figure 2.1: HDFS Architecture

is used to generate Block Token Authenticator (BTA) and Block Token. Block Token uses the following formats and function:

$$\texttt{BTID} = \{ \text{ED, KeyID, OID, BID, AM}\} \quad (1)$$
$$\texttt{BTA} = \text{HMAC-SHA1(Key, BTID)} \quad (2)$$
$$\texttt{BlockToken} = \{ \text{BTID, BTA}\} \quad (3)$$

Where ED is the expiration date for the Block Token, KeyID identifies the Key used in function (2), OID is the Block Token owner's ID, BID represents block's ID, AM defines which access mode Block Token has (e.g. read, write, copy and replace), BTID is Block Token ID, BTA stands for Block Token Authenticator, and HMAC-SHA1 represents a keyed-hash message authentication code which uses SHA-1 as its hash algorithm.

When accessing a file on HDFS, a process sends an access request to a Name Node. Once the request is authorized, the Name Node uses formats (1), (2) and (3) to generate a specific Block Token and sends the Block Token along with other necessary information (e.g. targeted Data Node location, etc.) back to the process which requested this access. Then, the process contacts the targeted Data Node and uses Simple Authentication and Security Layer (SASL/DIGEST-MD5) to authenticate itself. In this protocol, Block Token ID is used as Authorization ID, and Block Token Authenticator is used as Authentication Credentials. Once the Data Node receives the Block Token ID from the process, it computes the Block Token ID's Block Token Authenticator using function (2). The authenticator should be the same as the process already has if the token is valid. In SASL/DIGEST-MD5 protocol, the process and the Data Node should be able to authenticate each other by using their Block Token Authenticator as the shared secret key and DIGEST-MD5. After authentication is complete, the Data Node checks Block Token ID's validity, such as its expiration date, access mode, etc. If information inside the

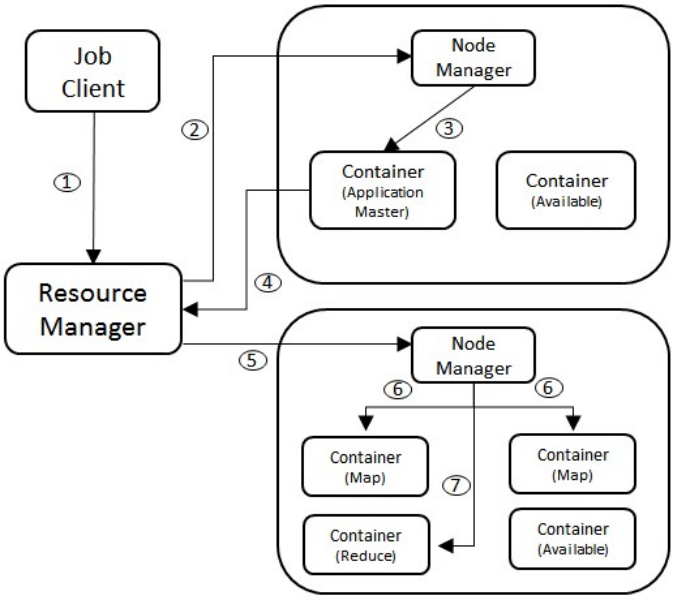

Figure 2.2: YARN Architecture

Block Token ID is correct, the Data Node performs data operations as the process requested. Because Block Token is designed to be lightweight and short-lived, there is no need to renew or revoke it.

2.1.2 YARN and Delegation Token

YARN is a framework for Hadoop job scheduling, monitoring, and resource management. It is known as MapReduce 2.0 (MRv2). YARN operates in a master/slave mode as described in Figure 2.2. Resource Manager (RM) is on the master computer. There is a Node Manager (NM) running on every slave computer. RM works on resource management and task scheduling. An NM is responsible for managing Containers assigned to it. A Container represents an amount of local computing resources. It runs as a Java process which is not operating-system-level virtualization Container. Each Container is used by an RM as an available unit to run applications, such as map processes, reduce processes, or Application Masters. An Application Master is created for a specific Hadoop job, and it works as the job's manager. It is responsible for communicating with the RM to get containers for executing the job code, tracking the status of these containers and monitoring the job's progress.

When a job request is received from a Hadoop user, in step 1 in Figure 2.2, the RM verifies the user's identity by using Kerberos and accepts the job if the request is authorized. Then the RM finds an available Container and negotiates with the Container's NM to start the specific

Application Master in steps 2 and 3. After that, the Application Master takes over this job. In step 4, the Application Master starts to send requests to the RM for more Containers to execute all the map's code and the reduce's code. Then in steps 5, 6 and 7, the RM locates additional Containers, and contacts proper Node Managers to create map processes and reduce processes. Finally, the Application Master keeps track of the state of all the Containers, and monitors the job's progress until the job finishes.

Delegation Token is used by map and reduce processes to authenticate themselves through a Name Node when they access HDFS. Delegation Token uses the following formats and function:

$$\texttt{DTID} = \{\text{OI, RI, IssueDate, MD, SN}\} \quad (4)$$
$$\texttt{DTA} = \text{HMAC-SHA1(Key, DTID)} \quad (5)$$
$$\texttt{DelegationToken} = \{\ \text{DTID, DTA}\} \quad (6)$$

Where OI represents Delegation Token Owner's ID, the owner of a Delegation Token is the Hadoop user who requested creation of the Delegation Token, RI is Renewer ID, which represents a Hadoop user's ID that can keep renewing the Delegation Token to keep it from expiring, IssueDate represents the time the Delegation Token is issued, MD stands for maximum date before the Delegation Token is expired, SN is a sequence number, DTID stands for Delegation Token ID, DTA represents Delegation Token Authenticator, HMAC-SHA1 represents a keyed-hash message authentication code which uses SHA-1 as its hash algorithm.

A Hadoop user employs a Job Client module to send commands to YARN and HDFS as shown in Figure 2.2. When a job request is received, the Job Client sends a Delegation Token request to a Name Node. Once the request is authorized, the Name Node generates a symmetric key, a Delegation Token ID, a Delegation Token Authenticator and a Delegation Token by using formats (4), (5) and (6). After that, the Delegation Token is passed to the Job Client through a secure channel. Then the Job Client and the Name Node share a Delegation Token Authenticator and a Delegation Token ID. The Job Client adds the Delegation Token to the job configuration file and submits the file with the job to Resource Manager. As mentioned previously, and in Figure 2.2, the Resource Manager creates an Application Master, maps processes and reduces processes for the job in sequence, and the Delegation Token is passed to these processes. When any of these processes need to access HDFS, the Name Node can use the Delegation Token to authenticate them. The SASL/DIGEST-MD5 protocol is used for authentication, and a Delegation Token ID is used as an Authorization ID, and a Delegation Token Authenticator is used as an Authentication Credential. After authentication, the Name Node checks Delegation Token ID's content to decide whether allowing the requested operation. If yes, the Name Node generates a Block Token and sends it back to the process. Then the process can use the Block

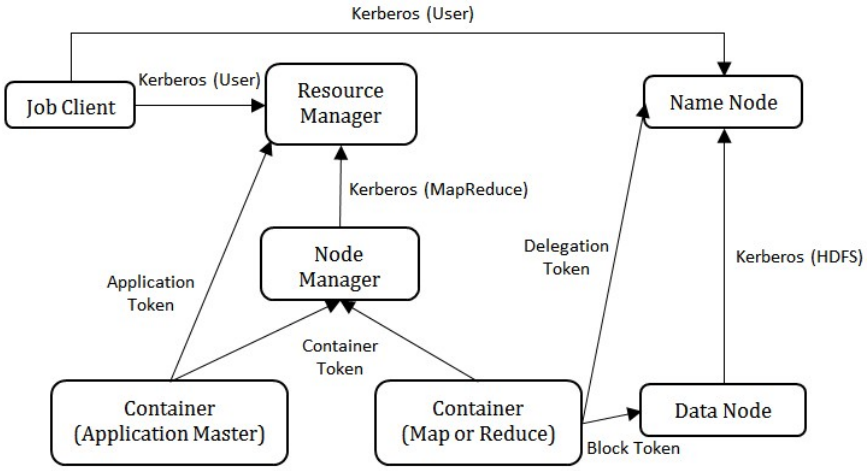

Figure 2.3: Hadoop Initial Authentication

Token to perform mutual authentication with the Data Node, and access data blocks on the Data Node. If not, the Name Node rejects this request from the process.

2.1.3 Initial Authentication

When Hadoop starts to run, it uses Kerberos for initial authentications as shown in Figure 2.3. In the YARN framework, Node Managers are authenticated by a Resource Manager through Kerberos. During initialization stage in HDFS, a Name Node uses Kerberos to authenticate each Data Node. When a Hadoop user needs to access YARN or HDFS, she or he uses Client to access both systems, and the Client is also authenticated by a Resource Manager and a Name Node through Kerberos authentication protocol. Data Node, Node Manager, and Client use secret keys or password to authenticate themselves to Kerberos. Data Node and Node Manager usually use secret keys. The secret keys are typically stored in node's local file system. Accidentally leaking these secret keys [22, 23] can enable attackers to be legitimate Hadoop component. Fortunately, each component of Hadoop has its unique secret key, so attackers can only impersonate the leaked key owner. However, this threat can combine with other vulnerabilities to compromise rest of Hadoop. We suggest that Hadoop administrators do not store these secret keys in OS images and that they safely clean the images before publishing Hadoop images in a cloud as mentioned in [23]. Hadoop administrator should bind these secret keys with fixed IPs, and update the keys regularly. All these methods can raise the bar and contribute towards preventing attackers from penetrating Hadoop.

2.2 The Threats to Hadoop in a Public Cloud

In a public cloud, multiple tenants share hardware and software resources. In IaaS model, if attackers share the same resources with Hadoop's VMs, they can launch various internal cloud attacks.

First, the threats can come from compromised hypervisors. Attackers can exploit vulnerabilities in hypervisors [17–20], and use methods demonstrated, for example by F. Rocha et al. [22], to obtain secret keys in Hadoop VM's file system and memory.

Second, the attacks can be launched from malicious VMs. In the work of T. Ristenpart et al. [24], they showed how to determine if two instances are running on the same physical machine, and launch side-channel attack in Amazon EC2.

Third, attackers can leverage the mistakes made by Hadoop administrators to start attacks. As demonstrated by S. Bugiel et al. [23], many users forget to delete secret keys in images before publishing their images in public. Hadoop administrators may forget to delete the secret keys which are used by Node Managers and Data Nodes to authenticate themselves from Kerberos before publishing Hadoop images. Hadoop uses Kerberos for initial authentication during initialization, and each Data Node and Node Manager uses its own secret key to authenticate itself. Attackers can use these keys to impersonate legitimate parts of Hadoop.

In a nutshell, while Hadoop is running in a public cloud, attackers can launch internal cloud attacks to bypassing current Hadoop security mechanisms, and steal sensitive information. One critical question is: when one or more Hadoop components are compromised, how much secure is the remain of Hadoop.

We evaluated security mechanisms in out-of-the-box Hadoop and identified two major vulnerabilities in two widely used authentication and authorization tokens, a Block Token and a Delegation Token.

Vulnerability 1: A Block Token is used by a process to access data on a Data Node. During HDFS initialization, a Name Node randomly generates a symmetric key and distributes it to Data Nodes when Data Nodes are booting up and contacting the Name Node for the first time registration. The key is used to generate Block Token Authenticator (BTA) and Block Token. A Name Node and all Data Nodes **share the same key** even in a large scale HDFS. If the key is leaked by any attack method, an attacker has the capability to use the key to generating arbitrary Block Tokens as he or she wants to, and accesses any data blocks in HDFS. The **overloaded authentication key** in Block Token dramatically weakens the isolation among components of HDFS.

Vulnerability 2: Delegation Token is used by map and reduce processes to authenticate themselves through a Name Node when they access HDFS. **The vulnerability is that Delegation Token lacks fine-grained access control.** With a Delegation Token, a process has

the privilege to behave as the Hadoop user and to access all contents which the Hadoop user is allowed to access. Lose of a non-expired Delegation Token can potentially enable the attacker to impersonate the Hadoop user and access the lost Delegation Token owner's data. A Delegation Token is used by all computing processes (i.e. map processes and reduce processes which are represented as Containers in Hadoop). These Containers are running across Hadoop processes. When Hadoop is running in a public cloud, Containers can also be the target of internal cloud attacks, and a leaked Delegation Token can be used to steal a large amount of data from HDFS. Therefore, fine-grained access control must be enforced on Hadoop computing processes to improve compromise resilience.

2.3 SEHadoop Model

2.3.1 Assumptions and Attack Models

Our goal is to improve Hadoop resilience to compromises. When compromises happen in parts of Hadoop, the security level of Hadoop should degrade gracefully, instead of losing all security protections. Let us assume that most of Hadoop components are running in a public cloud on IaaS model, and internal cloud attacks are possible. The attacks can be launched by using malicious hypervisors, co-resident VMs, mistakes of Hadoop administrators, etc. A secure zone can be created to shield critical parts of Hadoop from internal cloud attacks if administrators configure these critical Hadoop components correctly. We also assume that the secure cloud zone has dedicated hardware and software resources which have a strong isolation from other public cloud users. Amazon EC2 has dedicated instance services. Other well controlled private systems can also be used to host a secure zone. Since these services usually incur the higher cost than a public cloud does, we need to control the number of components running in a secure zone. We assume that attackers are only able to initiate limited scale internal cloud attacks on Hadoop components in a public cloud. In addition to security, we would like to maintain low operational overhead in Hadoop, and incur no extra storage overhead in HDFS.

2.3.2 Overview of SEHadoop

To improve Hadoop compromise resilience, we developed a new model we call SEHadoop using two major principles: enhancing isolation level among Hadoop components and giving least access privilege to Hadoop processes. When compromise starts in Hadoop, strong isolation level can help limit the extent of compromise. It forces hackers to attack one component at a time and this can slow down the pace of attacks. Enforcing least access privileges for Hadoop components can ensure that compromised Hadoop processes can only access limited data. Compared with original Hadoop, SEHadoop attackers need to attack more components to steal the same amount

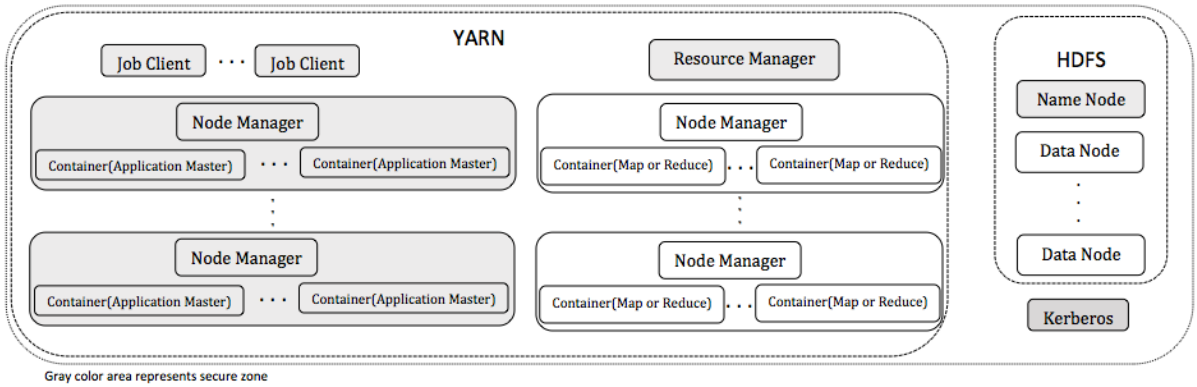

Figure 2.4: SEHadoop Runtime Model. Job Clients, Node Managers, Containers running Application Masters, Resource Manager, Name Node and Kerberos are in Secure Zone.

of data. Since some Hadoop components, such as Data Node, Node Manager, and Container, may be running on a large number of VMs, they have larger probability of being attacked than other Hadoop components. For example, a malicious VM of an attacker may have a better chance to co-reside with VMs running Data Node and to launch internal cloud attacks compared to VMs running Name Node. We carefully examined these components to ensure that the security mechanisms they are using satisfy the two principles. The SEHadoop model consists of SEHadoop runtime model, SEHadoop Block Token and SEHadoop Delegation Token.

2.3.3 Enhancing Isolation Level

Hadoop uses several secret keys and tokens to perform authentication and authorization among different Hadoop components. These keys and tokens may be used to attack Hadoop if they are leaked. In order to enhance isolation level, we want to minimize the number of shared keys and tokens. Block Token has a key shared with entire HDFS, and one Delegation Token is shared by all processes of one job. It is necessary to improve the security level of both tokens.

2.3.4 Least Access Privilege

Hadoop YARN processes data read from HDFS for computing. In out-of-the-box Hadoop design, each Container can access the entire data set of a user. In SEHadoop Delegation Token, we bring fine-grain access control to Containers and Node Managers, and each Container can only access a reduced data set it should be allowed.

2.3.5 SEHadoop Runtime Model

Some processes in Hadoop contain critical information. A Name Node manages the file system namespace and has the keys to generating Block Tokens and Delegation Tokens. An attacker can fetch all keys from its memory and access data from all active Data Nodes. A Resource Manager distributes all Delegation Tokens to proper Node Managers and Containers. These Delegation Tokens can be used by the hacker to access data of the Delegation Tokens' owners. An Application Master distributes the Delegation Token of one job to all the job's map and reduce processes. Once the keys or Delegation Tokens have been intercepted, an attacker can access a broad range of data in HDFS. A user uses a Job Client as an interface to access HDFS and YARN, the Job Client contains sensitive information (e.g. Kerberos' ticket and Delegation Tokens) and is able to access all the user's data. A Node Manager which manages Application Masters is responsible for setting up proper configuration, booting up the Application Master and transferring the Delegation Token to the Application Master. An attacker can use it to intercept all Delegation Tokens sent to an Application Master. Hadoop uses Kerberos to conduct most of the authentication operations. Compromising Kerberos, an attacker can impersonate any users in Hadoop. Therefore, a Name Node, a Resource Manager, Application Masters, Job Clients, Kerberos, and Node Managers running Application Masters should run in a secure zone. The rest of resources demanding processes of SEHadoop, such as Data Nodes, Node Managers, and Containers, run in a public cloud.

Fortunately, the scale of these processes is usually small or fixed. We propose a runtime model to protect these processes in a secure environment, named a secure zone, while other processes run in a public cloud environment. This is illustrated in the Figure 2.4. In a secure zone, a computer has dedicated resources to keep a public cloud threats away. In our model, we improve the overall Hadoop security level, while keeping most of Hadoop processes still in a public cloud environment. We minimize the number of SEHadoop processes running in a secure zone and the resource demands of SEHadoop in a secure zone. Storage processes (i.e. Data Nodes) and computing processes (i.e. Containers for map or reduce processes) usually have a great demand on resources, such as storage space, CPU, memory, etc., are all in a public cloud. In this model, we keep the need for a secure zone to the minimum, therefore, SEHadoop can enjoy the benefits of a public cloud, but also protect some critical Hadoop processes.

2.3.6 SEHadoop Block Token

We designed SEHadoop Block Token to fix the overloaded authentication key vulnerability. SEHadoop Block Token uses secret keys to conduct an authentication, but these keys are different for each Data Node. SEHadoop Block Token uses AES algorithm to encrypt plain-text of the token's content, thus SEHadoop Block Token's content remains protected during transfer from

a Name Node to a Data Node. The SEHadoop Block Token's generation formats are as follows:

$$\texttt{SEBTID} = \{ \text{ED, KeyId, UId, BPId, BId, AM, CIP}\} \quad (7)$$
$$\texttt{SEBT} = \text{AES(SEBTID, Hash(SEBTID))} \quad (8)$$

Where ED stands for expiration date of Block Token, KeyId is used to identify which key is used, UID stands for User ID, BPId represents block pool ID, BId defines which block needs to be operated on, AM means Access Mode (e.g. read, write, replace, copy) for this authorized operation, CIP is HDFS Client's IP. SEBTID stands for SEHadoop Block Token ID which defines all necessary information for an HDFS access request, Hash is the Secure Hash Algorithm (SHA256) hash function which is used to generate hash value of SEBTID, AES represents Advanced Encryption Standard encryption function which is used to encrypt the plaintext of a Block Token, and SEBT represents SEHadoop Block Token.

In SEHadoop model, a Name Node uses different keys to generate an SEHadoop Block Token for each Data Node. A Name Node creates a key set for all Data Nodes and manages the mappings from a key to a Data Node. Each Data Node has a unique symmetric key which is created by and shared with the Name Node. The keys are created when Data Nodes boot up and register with the Name Node. The symmetric keys are used in AES function in format (8) to generate SEHadoop Block Tokens.

In a Block Token of Hadoop, Block Token ID is transferred in plaintext from an HDFS Client to a Data Node. If such a Block Token ID is intercepted, its content becomes known to an attacker, and from this the attacker can deduce the topology of data in HDFS. SEHadoop Block Token uses AES and SHA256, which have low overhead, in format (8) to ensure SEHadoop Block Token ID's integrity and confidentiality.

A Data Node verifies an SEHadoop Block Token in two steps. First, it decrypts the SEHadoop Block Token using its secret key shared with the Name Node. The Data Node gets the SEHadoop Block Token ID and the hash value from the decrypted SEHadoop Block Token. Second, the Data Node computes a hash value of the SEHadoop Block Token ID and compares the result to the hash value from the decrypted SEHadoop Block Token. If both hash values match, this SEHadoop Block Token is verified. Then SEHadoop Block Token ID is used to authorize the access request.

2.3.7 SEHadoop Delegation Token

We designed SEHadoop Delegation Token to fix the lack of fine-grained access control vulnerability. SEHadoop Delegation Token must include fine-grained access control to allow a Name Node to restrict the content that the process can access when a Delegation Token is used. A Name

Node creates a Delegation Token but does not know the fine-grained access control information. Therefore, we propose to create SEHadoop Delegation Token in two steps. First, SEHadoop Delegation Token is created by a Name Node without fine-grained access control information. It carries data access authorization information and can be used to generate fine-grained access control SEHadoop Delegation Tokens in the next step. Second, a process, which has fine-grained access control information (e.g. Job Client), uses the SEHadoop Delegation Token to create multiple Delegation Tokens and adds fine-grained access control information into them. At the end, the Name Node must be able to verify all SEHadoop Delegation Tokens regardless of whether they are generated by the Name Node in the first step or by other processes in the second step.

SEHadoop Delegation Token consists of two parts: Parent Delegation Token and Child Delegation Token. There is one Parent Delegation Token and multiple Child Delegation Tokens for each job. A Parent Delegation Token is created by a Name Node. A Parent Delegation Token is used by a Job Client to generate multiple Child Delegation Tokens. Parent Delegation Token's formats is described as follows:

$$\texttt{PDTokenID} = \{ \text{OI, RI, IssueDate, MD, SN} \} \quad (9)$$
$$\texttt{PDTA} = \text{HMAC-SHA2(Key, PDTokenID)} \quad (10)$$
$$\texttt{PDToken} = \{ \text{PDTokenID, PDTA, CKey} \} \quad (11)$$

Where OI stands for the Parent Delegation Token owner's ID, RI means renewer's ID, IssueDate shows the date this Parent Delegation Token is created, MD represents the maximum date which is effective period of the Parent Delegation Token, SN is a sequence number for the Parent Delegation Token, PDTokenID is Parent Delegation Token ID, HMAC-SHA2 represents a keyed-hash message authentication code which the hash algorithm used is SHA-256, PDTA stands for Parent Delegation Token Authenticator, CKey stands for Child Key, PDToken is Parent Delegation Token.

Comparing to Hadoop Delegation Token, a Parent Delegation Token has a Child Key for generating Child Delegation Tokens in a Job Client later. The Name Node stores the Child Key while creating the Parent Delegation Token and uses the key to verify Child Delegation Token later. The proposed Child Delegation Token's formats are following:

Where Path is the input file path, StartOffset defines the offset of the first input byte in a file, Length is data input's length, CDTokenID stands for Child Delegation Token ID, PDTokenID means Parent Delegation Token ID, CKey represents Child Key, ChildDTA stands for Child Delegation Token Authenticator, HMAC-SHA2 represents a keyed-hash message authentication code which the hash used is SHA-256, CDelegationToken stands for Child Delegation Token.

`SplitInfo` = { Path, StartOffset, Length} (12)
`CDTokenID` = { PDTokenID, SplitInfo} (13)
`ChildDTA` = HMAC-SHA2(CKey, CDTokenID) (14)
`CDelegationToken` = { CDTokenID, ChildDTA} (15)

Child Delegation Tokens are created by a Job Client. A Job Client initializes a job and splits the input data for every map process. SplitInfo in format (12) defines the specific access control information, such as a file's path, first byte of offset in the file, and input data's length. A Parent Delegation Token ID and a SplitInfo comprise a Child Delegation Token ID in format (13). A Child Key is created by a Name Node and included in a Parent Delegation Token, so it can be used to generate a Child Delegation Token Authenticator in function (14). A Job Client can use a Child Key to generate Child Delegation Tokens without number limitation, and each map process can have its specific Child Delegation Token. These Child Delegation Tokens will be passed to a Resource Manager in a job submission and transferred to each map or reduce process. When a process uses its specific Child Delegation Token to access HDFS, the Child Token ID is sent to a Name Node. The Name Node uses the Parent Token ID which is inside the Child Token ID to search for which Parent Delegation Token this Child Token ID belongs to. The Name Node uses the Parent Delegation Token's Child Key to compute a new Child Delegation Token Authenticator based on Child Token ID received from the process. If the process has a correct Child Delegation Token, it must have the same Child Delegation Token Authenticator as the Name Node generated. Then the process and the Name Node have the same Child Delegation Token Authenticator as the secret and use SASL/DIGEST-MD5 protocol to finish authentication.

2.3.8 Security Analysis

To maintain a strong isolation level among components of Hadoop in a public cloud, Hadoop components should avoid sharing secret keys or tokens. Therefore, when one Hadoop component is compromised, its leaked keys and tokens will not expose keys and tokens of other Hadoop components. In SEHadoop, SEHadoop Block Token fixes the overloaded authentication key issue and resists attackers targeting storage processes (e.g. Data Nodes). Each Data Node is forced to share a unique symmetric key with the Name Node for generating SEHadoop Block Token, therefore, a compromised Data Node cannot leak keys owned by other Data Nodes, and attackers cannot access resources beyond the compromised Data Node. Each Container has a unique SEHadoop Child Delegation Token, and every SEHadoop Child Delegation Token can have different access privileges. A compromised Container does not have or has limited impact on data accessed by other Containers. By using unique keys and tokens, SEHadoop improves

the isolation level among components of Hadoop.

SEHadoop enforces least access privilege on Hadoop processes when these processes access data in HDFS. SEHadoop Delegation Token is used to perform authentication and authorization. SplitInfo in SEHadoop Child Delegation Token defines the fine-grained access control information. Each Container has a unique SEHadoop Child Delegation Token to restrict the range of data it can access. Thus, a compromised Container can only access the range of data permitted by a SEHadoop Child Delegation Token.

Since several Hadoop components (e.g. Name Node, Resource Manager) manage systemwide sensitive information (e.g. secret keys, user access control policies), compromising of these Hadoop components can cause a systemwide compromise. SEHadoop uses a secure zone to protect these components from various internal cloud attacks. The rest of components of Hadoop still run in a public cloud environment to acquire the benefit of cost saving.

By implementing SEHadoop, compromising small parts of Hadoop only has limited impact on the rest of components in Hadoop, therefore SEHadoop improves compromise resilience of Hadoop in a public cloud. However, if attackers can launch a large scale or systemwide attacks to Hadoop in a public cloud, e.g. compromising cloud management system, SEHadoop only has limited benefits.

2.4 Implementation and Experiments

In this section, we evaluate SEHadoop from three aspects: 1) SEHadoop security mechanisms are evaluated with two attack scenarios; 2) the complexity of migrating Hadoop jobs to SEHadoop; and 3) SEHadoop performance overhead is assessed.

2.4.1 SEHadoop Implementation

We implemented the SEHadoop model using Hadoop version 2.0.1-alpha. SEHadoop ran on virtual machines (VMs) for the experiments. The guest operating system was Ubuntu 11.04. All VMs ran on VMware ESXi 5.0.0. And VMware ESXi was installed on IBM eServer Blade Center HS21s with two Intel E5450 Xeon CPUs. Each CPU has 4 cores and the clock rate of 3 GHz. The network card was Intel Corporation 82545EM Gigabit Ethernet Controller, and network speed was 1 Gbit per second. Both SEHadoop and Hadoop were tested in the same cloud set-up.

We used seven VMs for the experiments with each VM having 7 GB available disk space and one CPU core. One VM was the master VM with 6 GB memory. Five VMs were slave VMs with 7 GB memory each. A separate 1 GB memory VM ran Kerberos. The last VM named Kerberos VM has 1 GB memory. The Name Node, the Resource Manager, and the Job Client

ran on the master VM. One Data Node and one Node Manager ran on each slave VM.

2.4.2 Security Evaluation

We developed two attack scenarios: the HDFS attack, and the YARN attack. Each attack scenario ran on both Hadoop and SEHadoop separately.

The HDFS attack tried to exploit the overloaded authentication key vulnerability and compromise HDFS. We demonstrated how one compromised Data Node can be used to attack the rest of components of HDFS and how SEHadoop prevents the attack. We assumed that one of the Data Nodes was compromised, and the malicious code was injected into that the compromised Data Node. The malicious code then got the Data Node's key and tried to directly read an HDFS block on another Data Node. The malicious code was in run() function inside BPServiceActor.java. It started to execute after the compromised Data Node finished the registration procedure with its Name Node and received the key. We assumed that the attacker knows the block's information, such as its location (i.e. the targeted Data Node's IP and port number.), Block Pool ID and Block ID. This information is stored in the Block Token Identifier and sent in plain-text through the network while a process using the Block Token to authenticate with any Data Nodes in Hadoop. One easy way for an attacker to fetch this information is to eavesdrop Hadoop's network. The attack code constructed the desired Block, then used the key shared by the Data Node and the Name Node to generate a Block Token. Finally, a Reader was created to read the whole block into the local buffer, and the content was displayed on the screen.

For Hadoop, our experimental results showed that an attacker can successfully fetch the targeted block's content without the Name Node detecting this malicious access. In our setup, the Name Node's IP was 10.2.0.30, the malicious Data Node's IP was 10.2.0.31, and the targeted Data Node's IP was 10.2.0.33. The following log from the targeted Data Node shows that the access request from the malicious code passed the access control mechanism on the targeted Data Node. The log displays that the targeted Data Node accepted the access request and sent 1032 bytes data from targeted Data Node which IP was 10.2.0.33 to the malicious Data Node which IP was 10.2.0.31.

```
13/10/12 20:53:20 INFO DataNode.clienttrace: src: /10.2.0.33:1004, dest:
    /10.2.0.31:40328, bytes: 1032, op: HDFS_READ, cliID: 10.2.0.33, offset: 0, srvID: DS
    -2118738703-10.2.0.33- 1004-1380768895735, blockid: BP-492552926-
    10.2.0.30-1380753088938:blk_14153627259595904_1663, duration: 205385
```

In the SEHadoop, the HDFS attack used the same method to fetch the content of the targeted block. But the malicious code could only use the key which the compromised Data

Node had. Since that key was different from the key used by the targeted Data Node, the attack failed. Because the AES keys used for the SEHadoop Block Tokens were different, the targeted Data Node rejected the malicious access request and threw out the bad padding exception when verifying the SEHadoop Block Token from the malicious Data Node. The following log trace on the targeted Data Node illustrates that:

```
javax.crypto.BadPaddingException: Given final block not properly padded
  at com.sun.crypto.provider.CipherCore.doFinal(CipherCore.java:811)
  ...
  at org.apache.hadoop.hdfs.security.token.block.BlockTokenSecretManager.
    decryptBlockToken(BlockTokenSecretManager.java:142)
  at org.apache.hadoop.hdfs.security.token.block.BlockPoolTokenSecretManager.checkAccess
    (BlockPoolTokenSecretManager.java:96)
  ...
```

The YARN attack aimed to exploit the lack of fine-grained access control vulnerability, and tried to leak data. We exhibited how SEHadoop Delegation Token used the least access privilege principle to restrict the range of data that a compromised Node Manager can access. A malicious Node Manager was used to intercept a Delegation Token, then the Delegation Token was used to access the Delegation Token owner's files. The malicious code was in the initialize() function of the LineRecordReader.java. When a map process was running on the malicious Node Manager, the malicious code was executed during the map initialization. Since the Node Manager controlled which Record Reader is used by a map process, the malicious Node Manager can ensure its map processes using the malicious Line Record Reader. There were two files in HDFS: test1 and test2. Both belonged to the Hadoop test user. When Hadoop Word Count job used test1 as an input file, the malicious code tried to read and download the entire content of both test1 and test2 files.

In the Hadoop, the malicious code successfully downloaded both files to the malicious Node Manager's local file system, which demonstrated that by owning a Delegation Token, a process can access any files owned by the Delegation Token owner.

On the other hand, in the SEHadoop, each Child Delegation Token contained SplitInfo which defining the file and data range allowed for access. When the malicious Node Manager used the Child Delegation Token to access the file which was not allowed in SplitInfo, such as the test2 file, the Name Node rejected the malicious access request and generated the access permission denied the exception. The log on the Name Node is illustrated as following, and the Name Node rejected the access for the file /user/hadoop/test2.

```
13/10/23 15:32:45 INFO mapreduce.Job: Task Id : attempt_1382556181136_0001_m_000000_0,
    Status : FAILED
Error: org.apache.hadoop.security.AccessControlException: Permission denied: user=hadoop
    /admin@hadoop.edu, access=Read, src=/user/hadoop/test2, SplitInfo is not Permitted
```

```
  at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.getBlockLocations(FSNamesystem.
    java:1142)
  ......
```

In the SEHadoop, the malicious Node Manager accessed the same file defined in the SplitInfo, but the range of the file was beyond the range defined by the SplitInfo, such as the entire content of the test1 file. The Name Node did not send the meta-data information (i.e. block location, block ID and SEHadoop Block Token) of blocks which was not defined in SplitInfo to the malicious client. Since the malicious client did not have meta-data information, especially the SEHadoop Block Token, of targeted blocks, the malicious access failed. In our attack scenario, the malicious code cannot find targeted block meta-data information and access failed. The log on the malicious Node Manager showed that the chooseDataNode function could not find the block meta-data for the targeted block, thus threw out a Block Missing Exception:

```
13/10/28 17:05:15 INFO mapreduce.Job: Task Id : attempt_1382993205375_0002_m_000004_0,
    Status : FAILED
Error: org.apache.hadoop.hdfs.BlockMissingException: Could not obtain block: BP
    -1130781010-10.2.0.30-1382376897952:blk_7510044259119580255_2085 file=/user/hadoop/
    test1
  at org.apache.hadoop.hdfs.DFSInputStream.chooseDataNode(DFSInputStream.java:657)
  at org.apache.hadoop.hdfs.DFSInputStream.blockSeekTo(DFSInputStream.java:437)
  at org.apache.hadoop.hdfs.DFSInputStream.read(DFSInputStream.java:587)
  ......
```

2.4.3 Migrating Hadoop Jobs to SEHadoop

All Hadoop jobs should be able to run on SEHadoop. Migrating Hadoop jobs to SEHadoop is straightforward. We divide jobs into two categories: splittable input job and non-splittable input job. A job is a splittable input job if the job's input can be divided into several consecutive pieces, named FileSplits, and each map process can handle one FileSplit. If not, a job is non-splittable. For instance, each map process of a job needs to access the entire input set.

Splittable input jobs can be migrated to SEHadoop by customizing a splitter to ensure that the splitter sets a correct boundary for each FileSplit. This enables SEHadoop to enforce access control, so each map can only access the content defined in its FileSplit. For example, Hadoop Word Count reads input data and counts how many times each word occurs. In the Word Count, each word is separated by a tab, a space, a newline character, or any other defined separator. However, the splitter only does logic splitting of input data, and the boundary of the FileSplit may be on the half of one word. In SEHadoop Word Count, we customize the splitter

and adjust each FileSplit size to ensure all boundaries of the FileSplit will not cut a word into a half. Then each map has proper the FileSplit range to access and is able to run on SEHadoop with fine-grained access control.

For non-splittable input jobs, we can customize the splitter to set the data range of each FileSplit to the entire input file. For these jobs, SEHadoop can provide file-level access control, but not have the block level fine-grained access control as a splittable input job has.

2.4.4 SEHadoop Performance

We compared Hadoop's performance with SEHadoop's performance in two cases. In the first case, we examined how much overhead SEHadoop Block Token incurs compared with Hadoop Block Token. In the second case, we evaluated the overhead of SEHadoop Delegation Token.

In the first case, in order to clearly compare the performance difference between Hadoop Block Token and SEHadoop Block Token, we minimized operation overhead except for token generation and token verification. We measured the time for a Client to read a small file, 1 byte in size, from both Hadoop HDFS and SEHadoop HDFS. The Client worked with the same Data Node in both Hadoop and SEHadoop scenarios. We used three VMs: Kerberos VM, master VM, and a slave VM. HDFS block replication is one. We repeated to read operation 10,000 times for each system. In the table 2.1, the average times and standard deviations for Hadoop and SEHadoop are showed in seconds. There did not appear to be statistically significant difference between two. The reason was that SEHadoop Block Token used low overhead algorithms, and it generated the same number of tokens as Hadoop did. If the block replication increases, SEHadoop Name Node has to generate one SEHadoop Block Token for each block replication while Hadoop Name Node only needs one Block Token for each block. So we measured the code operation time for SEHadoop Block Token generation and it took 1 to 2 ms in our experiments. SEHadoop Block Token generation time is very small compared with the overall reading time for one-byte reading. A large replication of blocks can dramatically reduce the available space of HDFS. The number of HDFS default replication is three. The SEHadoop Block Token generation time is less than 6 ms under the default replication, which is much smaller than the overall reading time. So SEHadoop Block Token performance overhead is still hard to observe in default replication scenario.

In the second case, we evaluated SEHadoop Delegation Token performance by measuring the time for generating Hadoop Delegation Tokens and SEHadoop Delegation Tokens. We excluded job computing time in the experiment to clearly display performance difference between two Delegation Tokens, due to job computing time varies a lot depending on job code and input data. We measured the time for a Job Client preparing the job and submitting the job to a Resource Manager. For Hadoop, this process includes requesting for a Parent Delegation Token

Hadoop	1.806 ± 0.024 (s)
SEHadoop	1.799 ± 0.033 (s)
Overhead Percentage	-0.0039%

Table 2.1: Hadoop and SEHadoop Block Token Performances

No. of Maps	80	640	5120
Hadoop	17.91 ± 0.07 (s)	17.99 ± 0.06 (s)	18.58 ± 0.16 (s)
SEHadoop	17.92 ± 0.09 (s)	18.36 ± 0.10 (s)	21.44 ± 0.28 (s)
Overhead Percentage	0.047%	2.0%	15.4%

Table 2.2: Hadoop and SEHadoop Delegation Token Performances

from a Name Node. For SEHadoop, the process includes requesting for a Parent Delegation Token from a Name Node and generating of Child Delegation Tokens, etc. The number of map determines how many Child Delegation Tokens need to be generated. We configured the job to generate a different number of maps in several cases to show how the number of maps impacts the performance of SEHadoop.

The randomTextWriter job was used as the performance test job. Table 2.2 shows how much time SEHadoop and Hadoop needed in seconds. The number of maps ranged from 80 to 5120. The SEHadoop overhead was comparatively small when there were 80 maps. But the overhead percentage became larger, such as 15.4%, when there were 5120 maps. Hadoop usually uses one map to process one block of data. Default block size is 128 MB. 5120 maps can have more than 655 GB input data. When there were 5120 maps, 2.86 seconds runtime difference between Hadoop Delegation Token and SEHadoop Delegation Token was a comparative small comparing to the overall job executing time to process 655 GB data which usually needs more than a half hour. The measured operations were running on the Job Client which is outside of SEHadoop computing cluster. So even for a Hadoop job with a large number of maps, SEHadoop Delegation Token has a very limited performance impact on overall Hadoop job executing progress.

Chapter 3

SafeDedup: Safely Deduplicating and Sharing Encrypted Data in a Cloud

3.1 Background

Cloud storage service prices are falling [50]. Some providers use deduplication to save space and reduce cost. Examples are Dropbox and Mozy [25, 51, 52]. While this may be good news for many cloud users, those who keep sensitive data in the cloud may need to worry. Deduplication requires reading the data and comparing files or blocks to know what to remove. That activity may expose sensitive information, or may not work if the user decides to use an arbitrary encryption approach not revealed to the deduplication process. Either way, there may be an increased cost.

As part of our work aimed at data protection in the cloud, we investigated the issue of deduplication of encrypted data. While there are several approaches in use today, they have some vulnerabilities. We developed a model we call SafeDedup, which we believe can solve the problem. We used Hadoop distribute file system (HDFS) as the case-study. There are three primary deduplication methods: deduplicating data based on whole file, fixed-sized chunk, and variable-sized chunk [53–56]. The model does not depend on file system organization (block, fixed or variable-sized chunks, whole file, etc.), and the deduplication method used. We first provide an overview of some existing approaches to the issue.

Convergent encryption is the popular solution to encrypted deduplication. It is deployed in many systems[25, 26, 33–35, 53, 57]. Since an encryption key is derived from the data by deterministic encryption (e.g. symmetric encryption or hash function), the same data can

generate identical ciphertexts which can be deduplicated. Mark Storer et al. use CE to develop a solution in both a single server storage and distributed storage systems, but the solution cannot prevent the brute-force attacks [53]. As mentioned in [25, 27], the brute-force attack can recover data sets which have low entropy, the so-called predictable data. Since predictable data is widely used in the real world, they proposed a separate key server for generating encryption keys to counter brute-force attack in Dupless [25]. However, a malicious insider can still use the key server to generate an encryption key and perform brute-force dictionary attack. This limits the adoption of this solution. Yitao improves Dupless by introducing a new distributed key generation protocol to remove the key server, so the method can be better adapted to storage systems which are not easily compatible with the centralized key server approach, such as P2P storage systems [57]. However if attackers can compromise a certain amount of users, Yitao's approach can not prevent attacks.

The model we developed, SafeDedup, takes the process one step further with the intent of countering such brute-force dictionary attack and resisting the threat of compromised users or malicious users. We leverage the isolation between virtual machines and the host machine which is supported by virtualization technology to process data in the host layer separated from the Cloud Storage Service (CSS) system. In this way, we separate sensitive information from storage application to thwart malicious inside users from accessing critical information, such as data encryption keys, ciphertexts of data, data sharing information, etc.

Another interesting and related problem is sharing of encrypted deduplicated data. Simply sharing a data encryption key does not satisfy compromise resilience requirement, and is not flexible enough if dynamic changing of access privilege for a shared party is needed. Yitao's distributed key generation protocol is able to support sharing of data within a group [57], however, the protocol lacks compromise resilience and fine-grain access control for each data share player. Our model improves on that with fine-grained access control, and enablement of the owner revoke of data sharing.

3.2 SafeDedup Threats Model

Our ultimate goal is to protect the confidentiality of users' data, even for the predictable data, both in deduplication enabled CSS and during sharing with other users or devices. We assume CSS resides with a trusted cloud provider, and all service processes are running in trusted virtual machines (VM) on hypervisors of the provider. Attackers may be able to compromise CSS, register as a user on CSS, or eavesdrop on connections between CSS and legitimate CSS users. We trust IaaS provider's management system, internal network, hypervisors, host machine hardware, etc.

When a compromise happens, the level of security should reduce incrementally and gracefully.

For example, if a client is compromised, the data of other users still remain safe. If CSS is compromised, all users data can still remain intact. If both CSS and IaaS providers' systems are compromised which is the worst scenario, data with unpredictable content are still safe, while predictable data may be under brute-force dictionary attack. In our model we divide data into different partitions. When the worst scenario happens, only data in compromised partitions are impacted.

Side-channels attacks also can be used to attack other users during client-side deduplication since a malicious user can tell whether one file is stored by others as described in [27]. SafeDedup model prevents these threats by hiding duplication information from the CSS system and users. Our model requests users to encrypt their data's path and file name if their path and file name are sensitive.

Our model provides a method to share encrypted data with fine-grain access control. A data owner is able to grant fine-grain access privilege to any other users or devices and perform revocation on any previous granting accesses. Compromise of CSS and clients does not impact uncompromised users.

Our approach works under CSS with minimum modification to the HDFS code. The space to store an encrypted file can match closely to the space of the plaintext. The deduplication function can achieve the same percentage space saving compared to the other systems [25, 26, 57].

3.3 SafeDedup Model

We describe SafeDedup from three perspectives: data and CSS separation, encryption and deduplication service, and data access protocol. Figure 3.1 is the proposed architecture of SafeDedup. It consists of Client, SafeDedup Hypervisor module, SafeDedup Service module, and SafeDedup VM module.

To separate CSS, SafeDedup leverages the hypervisor's privilege to intercept all data communications between CSS VMs and the clients. It ensures that all data are separated from the CSS system and remain in the hypervisor layer. At the same time, the CSS system can work properly and performs normal operations with the minimum CSS system modification. Data and CSS separation works with other parts of SafeDedup to ensure the security of the overall system.

In SafeDedup encryption and deduplication service, encryption and deduplication of data are performed securely in the hypervisor layer without conflicting. It works with the other two parts of SafeDedup to prevent attackers from leaking data. SafeDedup features compromise resistance when hackers are able to compromise critical parts of SafeDedup, such as SafeDedup Hypervisor module and SafeDedup Service module in the hypervisor layer. In our design, SafeDedup alone cannot decrypt the ciphertext, and users are necessarily involved in data decryption. SafeDedup

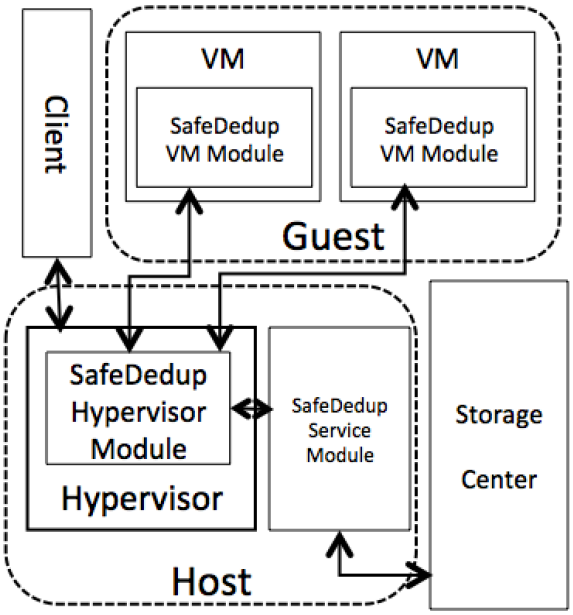

Figure 3.1: SafeDedup Architecture

has a mechanism that can partition data to effect compartmentalization.

Data access protocol is designed for communications among the CSS system components and clients. In addition to perform typical data operations, such as data read, write, delete, etc., the protocol is used by a client to confidentially share data with other designated clients using fine-grained data access controls. A data owner has the ability to set up access privileges for other users to define and enforce which data they can access, for what period of time and which operation privileges they have.

SafeDedup has two pairs of RSA asymmetric keys - the SafeDedup Public Key(SPub) and the SafeDedup Private Key (SPri), the SafeDedup Token Public Key (TPub) and the SafeDedup Token Private Key (TPri). Both SPri and TPri are only accessed by SafeDedup. There is a secure channel to distribute SPub and TPub to all clients, and clients are able to certify SPub and TPub in the same way public key infrastructure (PKI) does.

3.3.1 Data and CSS Separation

We assume that full virtualization is available. That should allow a host machine to provide completely virtualized hardware for guest virtual machines (VMs) which run an unmodified operating system (OS). Nowadays, major CPU manufacturers provide hardware-assisted virtualization to improve the performance of a guest environment, such as AMD-V [58] and Intel VT [59]. A hypervisor or virtual machine monitor (VMM) has higher privileges than all guest

VMs it runs. A hypervisor also enforces a strong isolation between guest VMs and the host environment. By leveraging the privilege of a hypervisor and the strong isolation between guest VMs and the host machine, SafeDedup separates data from the CSS system.

To do that, SafeDedup Hypervisor Module (see Figure 3.1) intercepts all incoming and outgoing network connections between CSS and the clients. Since CSS usually uses a pre-defined network port to receive and send data, SafeDedup Hypervisor module monitors all network connections on the predefined port. When a client sends data to CSS, SafeDedup detects the target connection, intercepts data in the channel and transfers data to SafeDedup Service module for encryption and deduplication. It then replaces data with a newly generated random number and puts it back onto the channel. The CSS system considers the random number generated by SafeDedup as data sent by the client. When a client tries to read the data, CSS sends the random number as the data to the client through the predefined port. SafeDedup Hypervisor module intercepts the random number from the monitored connection and uses the random number to retrieve real data from SafeDedup Service module. Then SafeDedup Hypervisor module replaces the random number with real data before sending packets to the client.

Because the CSS system decides how to perform operations on data (e.g. read, write, delete, etc.), SafeDedup needs information on which operations are performed on the target data. Then it can process the intercepted data accordingly. We use SafeDedup VM module in the CSS VMs to hook into the system call table. When the target operation is intercepted, such as writing operation in the target directory, SafeDedup VM module executes VMExit [59] to cause VM to exit, and SafeDedup Hypervisor module is able to identify which event (e.g. write or read) triggers the VMExit and other related information (e.g the random number CSS is writing). By using this approach, SafeDedup is aware of how the CSS system processes the random number and is able to process the real data accordingly. The CSS system does not know that the data is replaced by the random number, and performs the work as normal. The most important part is that the CSS system is not able to access the real data in all procedures.

A compromised CSS system can try to retrieve data in SafeDedup, such as working with malicious clients to perform illegitimate read operations on target files, or hack the hypervisor layer to leak data directly. Therefore, data and CSS separation that SafeDedup Hypervisor module effects is required to work with other components of SafeDedup to protect data.

3.3.2 Encryption and Deduplication

SafeDedup encryption and deduplication operations are implemented in SafeDedup Service module. SafeDedup Service module receives data and the commands (e.g. write) from SafeDedup Hypervisor module. Once a write command is received, SafeDedup Service module starts the data encryption procedure. The procedure has four goals:

- The same plaintext generates the same ciphertext, thus deduplication can be easily achieved.
- Only SafeDedup generates an encryption key for the data. Since the data encryption method is not publicly known, offline brute-force dictionary attack is much more difficult unless SafeDedup Service module is also compromised.
- SafeDedup is able to partition data. Data in different partitions is encrypted by different sets of encryption keys. Deduplication can only work for data in the same partition. SafeDedup can partition data according to either different organizations or time periods to improve compromising resilience.
- Only the data owner is able to decrypt his or her ciphertext. Without authorization by a data owner, a client or a compromised SafeDedup can not decrypt the ciphertext.

When SafeDedup Hypervisor module intercepts a data write operation on the CSS system, plaintext of data, the random number, and related information are sent to SafeDedup Service module. SafeDedup Service module uses plaintext and SafeDedup master key (MK) to generate a symmetric data encryption key DK $\leftarrow$ F(MK, plaintext). We use HMAC as the function F to generate DK. Then DK is used to encrypt data into ciphertext C $\leftarrow$ E(DK, Plaintext). When the plaintext and the MK are the same, SafeDedup ensures that the same DK and ciphertext are generated. When the data or MKs are different, SafeDedup generates different ciphertexts. Thus, MK can be used to partition data for various proposes.

In order to ensure only data owner can decrypt the ciphertext, SafeDedup requires a data owner to generate a random symmetric key, called tag key (TK), and send the key with data to SafeDedup. SafeDedup Hypervisor module intercepts both TK and data, and sends them to SafeDedup Service module. TK is used to create a tag through Tag $\leftarrow$ E(TK, DK || H(DK)) which H is a Hash function. At last, the tag, the random number, the ciphertext and the ciphertext ID which is the index of ciphertexts are stored. At the same time, a mapping of the tag, the random number, and the ciphertext ID is recorded as described in the table 3.1, therefore, SafeDedup can identify which tag and ciphertext should be used when a read request is received. Having the same ciphertext ID in the rows of the tag A and the tag C indicates that both tags share the same ciphertext. The data owner who has the TK is able to decrypt the tag to retrieve the DK which can be used to decrypt the ciphertext. In our implementation, we use access token to authorize SafeDedup to perform decryption which is discussed in Section 3.3.3.

3.3.3 Data Access Protocol

SafeDedup data access protocol defines the communication procedures among clients, CSS and SafeDedup. It provides three functions: a secure channel is created for communications between a

Tag	Random Number	Ciphertext ID
Tag A	Number D	X
Tag B	Number E	Y
Tag C	Number F	X
...	...	...

Table 3.1: Tag, Random number and Ciphertext mapping

client and CSS, a user can use a client to access data (e.g. write or read data) in SafeDedup and CSS, a data owner can share data with others and control their access in a fine-grain manner.

Secure Channel

Since a client is usually installed on users' device which remote to the CSS servers, such as a computer or a tablet, the communication between a client and the CSS system normally travels through Internet which does not have security guarantee. The secure channel between a client and the CSS system is needed to protect the communication between them. SafeDedup has to be able to intercept the related information and data in this channel. Because SafeDedup Hypervisor module is running in the hypervisor layer, and the communication between a CSS VM and its hypervisor is actually memory page copying on the host machines which is hard to be intercepted by external attackers. We can leverage this fact, and only create an encrypted channel between SafeDedup Hypervisor module and a client to simplify our data access protocol.

When a client needs to send a request to the CSS system (e.g read or write), it creates an random AES256 symmetric key, named session key (SK), to encrypt the entire request (R) via ER $\leftarrow$ E(SK, R || H(R)), which ER is encrypted request and H is cryptographic hash function SHA-512. SK is encrypted by SPub in the head via Head $\leftarrow$ E(SPub, SK || H(SK)). Once SafeDedup Hypervisor module intercepting Head and ER, it decrypts Head by using its SPri, and verifies the integrity of SK by checking the hash value of SK. Then SafeDedup Hypervisor module uses the SK to decrypt ER and verifies the integrity of R via calculating the hash value of R. The SK received by SafeDedup can be used to encrypt the CSS system response (SRes) in the same way as how the client encrypting request via ERes $\leftarrow$ E(SK, R || H(SRes)). Once receiving encrypted response from SafeDedup, the client uses its SK to decrypt it and verify its integrity. The Figure 3.2 describes the process of the secure channel. SD represents SafeDedup swapping data operation which swaps the data with the random number in the network buffer of the hypervisor as described in Section 3.3.1. A secure channel between a client and the CSS system is established and all communications between clients and the CSS system are transmitted through it.

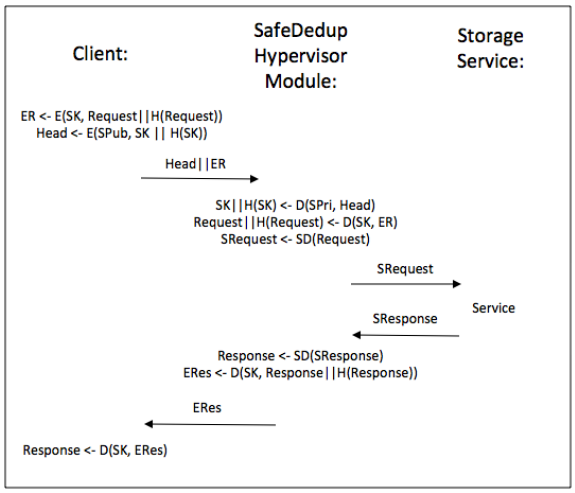

Figure 3.2: SafeDedup Secure Channel

Data Access

An access token (AT) is used to authenticate and authorize access in SafeDedup. A tag key (TK) is generated when the data is written to SafeDedup as described in 3.3.2, and each block of data has its own TK. An owner of the TK can use the TK and TPub to generate access tokens. These tokens can be used by the owner or any third parties to access data in SafeDedup accordingly. Inside access tokens, it defines fine-grain access control information, including the type of operations (e.g. read, write or delete), expiration date, etc. If the data owner needs to share data on the CSS system, she or he only needs to distribute access tokens to any third-parties instead of exposing decryption keys. The format of an access token is described as follows:

$$\texttt{TID} = \{\text{ID, Operation, Expiration Date, Tag Key}\} \quad (16)$$
$$\texttt{AT} \leftarrow \text{E(TPub, TID } || \text{ H(TID))} \quad (17)$$

While writing new data into the CSS system, a client creates a random symmetric key TK to generate an access token (AT) via formats (16) and (17), and attaches AT and data to the request. The client sends the request to SafeDedup and the CSS system through the secure channel which described in Section 3.3.3. SafeDedup decrypts the access token by D(TPri, AT) and verifies the hash value of TID. If the Operation in TID is written, the Expiration Date is not exceeded and the CSS system allows the write operation, SafeDedup uses the TK to performs encryption and deduplication of data as discussed in Section 3.3.2.

In a data read procedure, a client creates an access token or directly uses an access token from the data owner to access data. The access token generation method is described in formats (16) and (17). The AT is sent to SafeDedup through the secure channel and decrypted by TPri. If the TID is verified, the Operation in TID is read, the Expiration Date is not expired, Tag Key is verified and the CSS system allows the read operation, SafeDedup performs decryption operation and sends the data encrypted by the SK back through the secure channel. Since each tag is created by Tag $\leftarrow$ E(TK, DK || H(DK)), a tag can be decrypted by DK $\leftarrow$ D(TK, Tag) to retrieve DK. The hash value of DK can be verified by recomputing H(DK). If the hash value of DK is verified, then both the DK and the tag key are correct.

In data read and write processes, SafeDedup adds another layer of authentication and authorization above the CSS system . A user can read and write data only if she or he can be authenticated and authorized by both the CSS system and SafeDedup model. Therefore, SafeDedup has the high privilege to enforce access control for users accessing the CSS system. SafeDedup access token is encrypted by TPub, and only SafeDedup has TPri to decrypt the access token. Thus, the access control information and the tag key in an access token are not exposed to any third parties and are not able to be modified by others.

3.3.4 Data Sharing and Access Revocation

SafeDedup can use an access token in the same way as discussed in Section 3.3.3 to share data with others. A data owner is able to create an access token by the formats (16) and (17) which contains the fine-grain detail of access privilege that the data sharing parties can have. We assume that the access token can be distributed to all data sharing users in a secure method. SafeDedup can use the same way to verify data access requests from different data sharing users by examining access tokens that the users provided. The details of access token verification are stated in Section 3.3.3.

SafeDedup has flexible ways to revoke an access token before the access token expiration date. The first revocation method is to keep a black list in SafeDedup which contains all unexpired revocation access tokens. SafeDedup will reject access requests which use the access tokens on the list. The expired access token will be removed from the list to keep the access token black list efficient. The second revocation way is that a data owner can replace the tag for her or his data. This method is very necessary when a data owner worries her or his tag key is leaked. Both maintaining access token black list and replacing a tag key are quite lightweight operations. Both an access token and a tag in SafeDedup have small sizes. Access token only contains keys and access control information. And SafeDedup tag only includes one key. These two methods are lightweight operations and do not involve any operations on users' data.

3.4 Security of SafeDedup

The evaluation of SafeDedup model is intended to demonstrate that the data in the CSS system is secure under a variety of foreseeable scenarios. First, we examine how external adversaries can affect SafeDedup. Second, we discuss possible data leaks from CSS system and cloud providers.

3.4.1 External Adversaries

A secure CSS system must be able to prevent information leaking to external adversaries. An external adversary does not have an account on the CSS system. While trying to access data on the CSS system, the access request of the external adversary will be denied by the CSS access control system. Communications between the CSS system and SafeDedup are operated directly on the memory of host machines which is not accessible for external adversaries.

However, an adversary is able to overhear and tamper with messages sent between legitimate CSS users and the CSS system. In our model, all messages between CSS clients and the CSS system are transmitted in a secure channel. Messages which are transmitted through the secure channel are encrypted. We assume that encryption techniques provide an adequate level of security over a certain period of time. A sender generates a hash value of its message and sends it with the message. A receiver can then verify the message's integrity by recomputing the hash value of the message.

3.4.2 Internal Adversaries

We consider three categories of internal adversaries. The first is an adversary that compromises client devices. The second is CSS internal adversary who has the privilege to access the CSS system by either using a normal user account or an administrator account on the CSS system. The third is cloud internal adversary who has the privilege to access the same cloud the CSS using. The adversary may have a user account or has an administrator privilege in the cloud. We analyze the ability of inside attackers based on their locations and access privileges in the system.

Client Internal Adversaries

In our model, we assume that users only use access tokens on their devices or computers to access data, and their tag keys are safely and separately stored to lower possibility of leaking the keys. Therefore, a compromised computer can only leak access tokens stored on the computer instead of tag keys. Attackers can only have the access privilege defined in the leaked access tokens. And the legitimate user can revoke the compromised access tokens while detecting the leaking event. It is users' responsibility to safely use and store tag keys. If tag keys are leaked,

attackers must pass access control of the CSS system in order to read data from SafeDedup. SafeDedup can send alerts of potential tag keys to users when an access token is verified but the access request is rejected by the CSS system. And the revocation process can be used to replace leaked tag keys.

CSS Internal Adversaries

A CSS internal adversary who has a user account can perform normal read, write, delete operations as other legitimate users. A CSS internal adversary with an administrator privilege on the CSS is able to access and manage resources of other legitimate users. The goal of SafeDedup is preventing both predictable and unpredictable information leaking to CSS internal adversaries. Although convergent encryption provides protection for unpredictable data [25, 32, 60], as discussed in [27], the CSS malicious users can perform brute-force dictionary attacks to leak predictable data even when the data are encrypted. This type of attacks is possible based on following three facts. Firstly, attackers can detect whether their uploaded files are shared by other legitimate users. Secondly, they know the ciphertext of the target file. At last, they are able to perform data encryption operation either by themselves [27] or by the CSS system [25]. It is hard to prevent CSS internal adversaries if CSS uses client-side encryption and deduplication. Thus, SafeDedup performs encryption and deduplication on server-side. We discuss how SafeDedup prevents data leaking in following.

As discussed in Section 3.3.1, the separation between CSS and data ensures that the CSS system cannot access both plaintext and ciphertext of data. Ciphertexts of all data are managed and stored by SafeDedup, and the CSS system is not able to access SafeDedup in the hypervisor layer. However, a malicious CSS administrator who manages the access control system of the CSS system may try to impersonate a legitimate CSS user to access data in SafeDedup. SafeDedup requires a valid access token to authorize a read operation, and only data owner has the tag key to generate the access token. Therefore without related tag keys or access tokens, CSS internal adversaries cannot directly retrieve plaintext or ciphertext of data from SafeDedup.

A master key (MK) is required to generate data encryption keys and is only stored in SafeDedup. Therefore, data encryption keys can only be generated by SafeDedup, and the data encryption process is not known by CSS internal adversaries. Thus, the off-line brute-force dictionary attack cannot be performed. On the other hand, CSS internal adversaries can request SafeDedup to execute data encryption as normal CSS users to launch online brute-force dictionary attack. We can stop this type of attacks by preventing attackers from detecting whether their uploaded files are shared with other users. Since SafeDedup uses server-side encryption and deduplication, attack methods in [25, 27] cannot leak data sharing information from the CSS system. However, other types of side-channel attacks are possible to leak file sharing information.

For instance, an attacker can write a file to CSS as a normal CSS user and read the file back immediately. If the response time of the read request is short that means the uploaded file is shared with other users and SafeDedup skips the operation of writing the file to a disk, and vice versa. But this side-channel attack can be easily defeated by setting up a delay timer in SafeDedup to restrict the minimum time interval between writing and reading operations, so even a uploaded file is shared on the CSS, the read request is delayed by the timer to avoid the file sharing detection. The length of delay time depends on the size of a uploaded file, the larger the file is, the longer for the time delaying. In the future, we can study about how to add extra randomness to the delay time to avoid attackers detecting whether the delay of reading request is caused by a delay timer.

Cloud Internal Adversaries

Cloud internal adversaries are able to access the cloud system and have a high privilege to launch various attacks directly on the CSS system and SafeDedup. As discussed in [3, 15, 16, 22, 24, 61], internal cloud attacks can be used to attack cloud, such as scanning or modifying the memory of the CSS' VMs, implementing malware into the images CSS using, etc. Fortunately, in SafeDedup model, the CSS system does not contain users' data or is able to access to these data alone which are analyzed in Section 3.4.2. However, cloud internal adversaries with high privilege can compromise SafeDedup directly. Since all data is encrypted in storage, and only data owners have tag keys to retrieve data decryption keys in tags, compromised SafeDedup alone cannot decrypt ciphertexts of data, especially for unpredictable data. But brute-force dictionary attack is possible for attacking predictable data in storage. SafeDedup improves the security level for these predictable data by using different master keys.

Firstly, we can partition data into different time slots. SafeDedup can refresh the master key for certain period of time and the old master key is always deleted. Therefore, the files written to SafeDedup in the different time period are stored in separate partitions. Compromising SafeDedup can only impact data which is written in a current period of time. Since lacking old master keys, attackers cannot repeat the data encryption key generation process to launch brute-force dictionary attack. This partitioning method can protect predictable and unpredictable data in old partitions and only protect unpredictable data in the current partition. The tradeoff for this method is that it sacrifices some benefits of space saving of deduplication. Setting up a proper length of time for SafeDedup refreshing a master key provides a flexible way for an administrator to control the balance between the data protection and storage saving.

Secondly, SafeDedup can use a unique master key for each cloud site. Therefore compromising of one cloud site does not impact data stored by SafeDedup in another cloud site. One scenario for this method is that a CSS system runs on multiple clouds, and SafeDedup modules in

different clouds use distinct master keys. Using a unique master key in each cloud or cloud site can prevent cloud internal adversaries launching brute-force dictionary attack on data encrypted by uncompromised cloud or cloud sites.

3.5 Implementation and Performance

We implemented fully functional SafeDedup. It is written in C and supports Hadoop Distributed File System (HDFS). It can be straightforward to make SafeDedup work with other distributed file system which is similar to the ways to support HDFS as described in Figure 3.1. We use Kernel Based Virtual Machine (KVM) 3.8 and QEMU 1.4.0 as the hypervisor in our platform. SafeDedup Hypervisor module is implemented in KVM and QEMU. We use CentOS 6.6 64-bits with kernel version 3.8.1 as the operating system (OS) in the host machine. SafeDedup Service module is running as a user process in the host machine. Our guest VM use Ubuntu 13.04 32-bits with kernel version 3.8.0-19-generic. Hadoop Distribute File System (HDFS) 2.2.0 is used as our distribute file system running on the guest machine. We use CentOS 6.6 64-bits as our client machine. HDFS client and SafeDedup client are used to read or write data in HDFS. We use SHA-512 hash function to calculate all hash values in SafeDedup. Advanced Encryption Standard (AES) 256 is used for all symmetric encryptions, and RSA (2048 bits key) is used for all asymmetric encryptions in SafeDedup.

SafeDedup Hypervisor module is implemented in QEMU, KVM and HDFS client. QEMU is an open source hosted hypervisor that works with KVM kernel module to perform hardware virtualization [62, 63]. QEMU provides the virtual network devices to guest VMs and interacts with these network interface controllers (NIC). The part of SafeDedup Hypervisor module is implemented in QEMU to monitor TCP/IP connections of all VMs, intercept the connection and replace the data in the network buffer for the connection. The other part of SafeDedup Hypervisor module is installed in KVM to interact with SafeDedup VM module in VM. SafeDedup VM module is a kernel module installed in guest VMs. It modifies the system call table and hooks four system calls which are open, read, write and close. When guest VMs try to read or write the target file, the module executes VMCALL which causes VMExit. Then SafeDedup Hypervisor module component in KVM can intercept the events, handle the exceptions and read information, such as which instruction caused VMExit and what value is read or written. We keep the modification of the CSS system to a minimum, and remove the two lines of code for the checksum verification in HDFS Data Node for uploaded data from HDFS clients. While writing data to the CSS system, the SafeDedup client generates the request message including the access token and data and encrypts the request as described in Section 3.3.3. The SafeDedup client packs all contents of a request to a local file for HDFS client to upload to HDFS server.

We described and discussed the experiments we tested in the SafeDedup platform, and

measured the performance and overheads of SafeDedup.

3.5.1 Experiments

We measured the overhead for both write and read operations on HDFS platform, and HDFS with SafeDedup platform. Both platforms use the same machines for the server and client. Measurements were repeated 100 times and all standard deviation are below 5%. The files used in the operations have random contents and their sizes are 2^{4i} KB for $i \in \{0, 1, 2, 3, 4\}$, giving a file size range of 1KB to 64 MB. We use the following function to calculate SafeDedup overhead percentage (A is native HDFS operating time, and B is HDFS with SafeDedup operating time):

$$\frac{B - A}{A} \times 100\%$$

The SafeDedup host machine is an IBM eServer Blade HS22 with two Intel E5640 Xeon CPUs. Each CPU has 4 cores and the clock rate of 2660 Hz. The memory size is 32 GB. The network card is Broadcom Corporation NetXtreme II BCM5709S Gigabit Ethernet, and network speed is 1Gbit per second. The SafeDedup client machine is an IBM eServer Blade Center HS21s with two Intel E5450 Xeon CPUs. Each CPU has 4 cores and the clock rate of 3 GHz. The memory size is 32 GB. The network card was Broadcom Corporation NetXtreme II BCM5708S Gigabit Ethernet controller, and network line speed was 1 Gbit per second. Both SafeDedup host machine and client machine use the same model of the hard disk which is Fujitsu MBD2300RC with 300GB, 10000 RPM, 16MB Cache and SAS 6Gb/s.The SafeDedup host machine and client machine are attached to the same VLAN, the average transferring bandwidth between two machines is 335 Mbits per second. We use one for the number of the replication, and 128 MB for the block size in the HDFS.

3.5.2 SafeDedup Performance

We performed write operation under two platforms, HDFS, and HDFS with SafeDedup. The results are shown in Figure 3.3. The content written to HDFS is random. HDFS itself does not perform deduplication. Therefore, there is no performance difference between writing the same data and different data. Since SafeDedup performing deduplication, we tested SafeDedup write performances by writing the same data or different random data. From Figure 3.3, a client cannot observe the performance difference when different data or the same data is written to SafeDedup. The reason is that SafeDedup service module performs deduplication and data storing in the background which does not influence writing performance from SafeDedup client perspective. That is extremely important because attackers cannot detect whether a file is stored in SafeDedup by other users through measuring its writing performance. Thus the side-channel

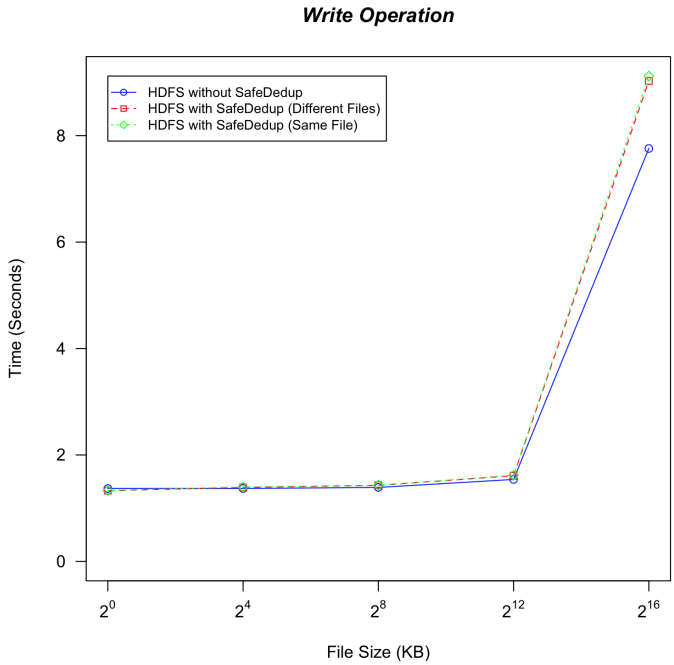

Figure 3.3: Write Operation Performances

attack in [27] cannot be launched as discussed in Section 3.4.2. Comparing HDFS and HDFS with SafeDedup, we cannot observe overhead for HDFS with SafeDedup when uploading a 1 KB file. We think that the overhead for encryption and decryption operations of SafeDedup, and data copy in the hypervisor is too small compared with operation time for data transferring over network and data writing on the hard disk. The overhead of SafeDedup starts to rise to 16.4% when the file size increases to 64 MB.

We compared read operation performances between HDFS and HDFS with SafeDedup as described in Figure 3.4. Since HDFS is not modified, and SafeDedup Access Token has to be uploaded to SafeDedup, SafeDedup Client performs two TCP connections with HDFS for one read operation. One connection is uploading SafeDedup Access Token and the other one is downloading data. On a contrary, HDFS only needs one TCP connection for downloading data. In order to improve the performance of SafeDedup, we use two threads to perform these two operations in parallel. Figure 3.4 indicates that SafeDedup has 8.7% overhead compared with HDFS when data size is 1 KB. The overhead percentage starts to increase to 18.8% when the data size is 64 MB.

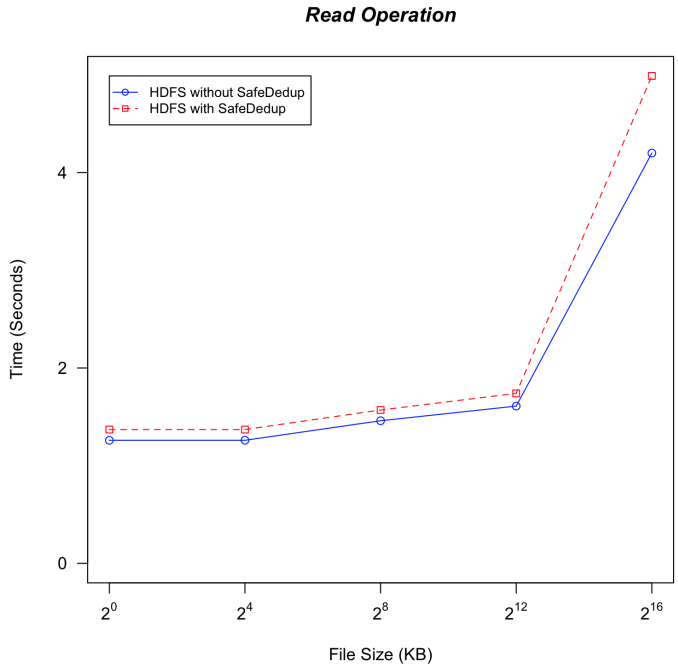

Figure 3.4: Read Operation Performances

3.5.3 Microbenchmarks

We measured the micro-benchmarks for SafeDedup Service Module performances as shown in Figure 3.5, which indicates the overhead for data encryption and decryption, data integrity check, data encryption key generation, data storing and reading, etc. When Safededup receives writing requests, SafeDedup Service Module will decrypt the data and the access token from the SafeDedup secure channel, verify the integrity of the data and the access token, generate a data encryption key for the data, encrypt the data and store the data in hard disk. The write line in Figure 3.5 shows the time to perform these operations under the different sizes of the random files. It requires from 0.013 seconds for 1 KB data to 4.05 seconds for 64 MB data to encrypt and store the data on the disk.

When reading requests are received, SafeDedup Service Module will read the encrypted data from the hard disk, decrypt the data encryption key from the Tag and use the data encryption key to decrypt the data. The read line in Figure 3.5 shows the time to perform these operations under the different sizes of the random files. It needs from 0.0023 seconds for 1 KB data to 1.35 seconds for 64 MB data to read and decrypt data on the disk.

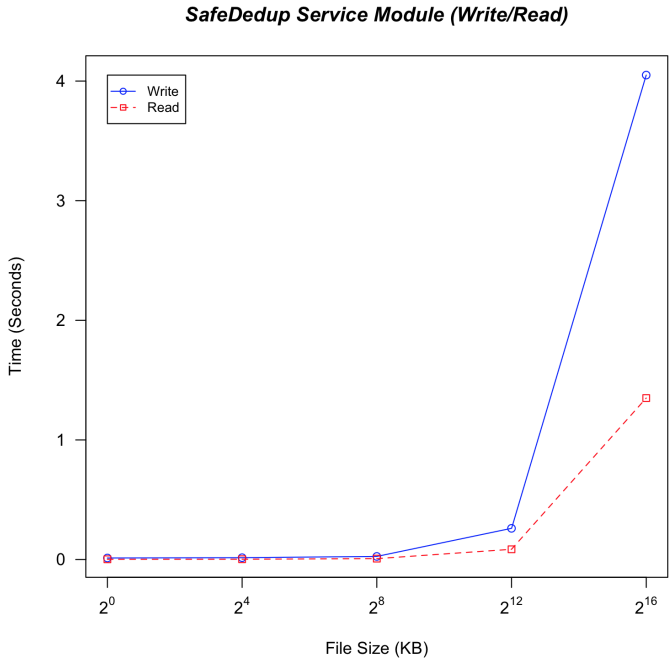

Figure 3.5: SafeDedup Service Module Performances

Chapter 4

SECross: Securing Cross Cloud Boundary

4.1 Background

Working across several clouds may represent a security challenge. We examined some of the issues experimentally. To do that, we used three existing cloud systems. We used North Carolina State University (NCSU) Virtual Computing Lab (VCL) as our base cloud system, and designed and implemented two deployer modules to enable VCL deploying and management capabilities in IBM Softlayer Cloud and Amazon Elastic Compute Cloud (EC2).

4.1.1 VCL Background

Virtual Computing Lab (VCL) is an open source cloud computing management platform and primarily used by colleges and universities to deliver cloud service to students, staffs and faculty. VCL can deliver infrastructure as a service (IaaS), such as bare-metal machines or virtual machines with productivity software, to users. The users can connect to the remotely provisioned computers by using remote desktop, SSH or other supported protocols.

VCL Architecture

VCL has four principal components: web portal, database, management node VCL daemon and deployer module as shown in Figure 4.1. It also has a number of distributed computational and storage resources from which the VCL management nodes deliver services. Web portal is a web interface for end users and administrators. VCL users use the web interface to send requests, manage resources. VCL administrators use the web interface to manage resources and set up configurations. VCL database stores all data of VCL reservation, environment configurations,

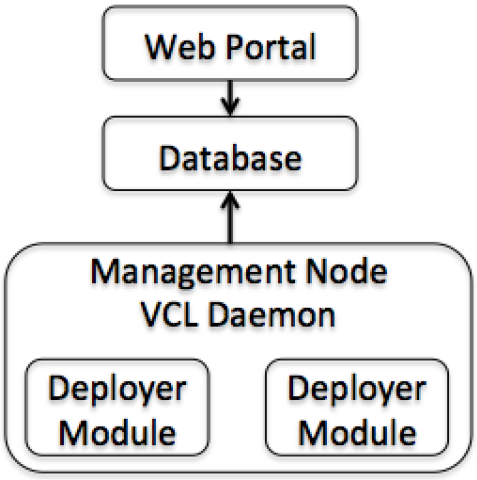

Figure 4.1: VCL Management Architecture

access control policies, log, etc. Management node processes reservation requests and uses deployer modules to load proper resource machines for end users.

Using VCL

In general, users log in to VCL through web portal. However, it is also possible to request and deploy services using an API. There are three methods can be used to authenticate users, VCL local accounts and their passwords, Lightweight Directory Access Protocol (LDAP) and Shibboleth.

After authentication, VCL has a privilege hierarchy to manage access privileges for all users. Users can manage resources, such as computing resources and image resources, or send requests, such as provisioning computing environment and creating new images, which are stored in VCL database in the end. Management Node keeps checking if there are any computing resources requests in VCL database.

If a new request is detected by Management Node, it starts to retrieve related information of the request from the database and call a proper deployer module to provision the requested computing environment.

Then the VCL deployer provisions the image, sets up network environment, creates the user's account and configures the computer according to the provisioning request. Users can log in to the computing environment by different methods, such as Secure Shell (SSH) or Remote Desktop Protocol (RDP).

When users finish work, they can end the reservation directly or send update image or create a new image request to store the current image.

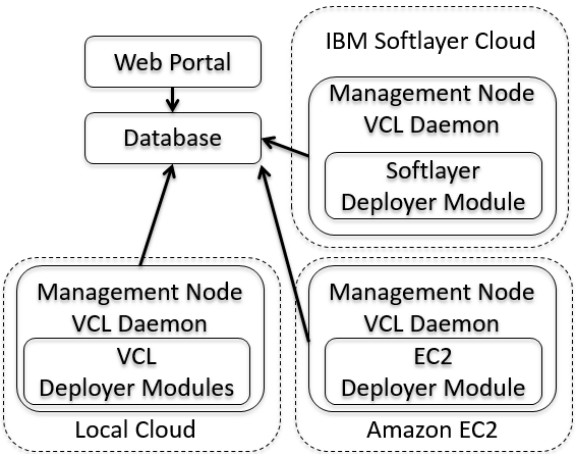

Figure 4.2: Multi-cloud VCL System

4.1.2 VCL Multi-cloud system

We developed two deployer modules which are Softlayer deployer module and EC2 deployer module. As shown in Figure 4.2, VCL multi-cloud system can have multiple management nodes, and each management node runs in different clouds environment, including local cloud which is fully controlled and managed by VCL administrators, and public clouds such as Softlayer cloud and Amazon EC2 which are controlled and managed by independent providers. The management nodes in public clouds can use these two deployer modules to communicate with public cloud management systems. Therefore, the management nodes can reserve and delete computing resources requested by VCL users.

4.2 Assumptions and Attack Models

We built an experimental multi-cloud system that has a private cloud computing environment as its base-line, while the other cloud environments are the public clouds (Amazon EC2 and IBM Softlayer in our case) are managed and controlled by independent cloud providers, and multi-cloud administrators do not have administrator privilege in these cloud environments.

We assume that a public cloud is by definition open to public users, malicious users can easily impersonate normal users. Multi-cloud system components in the public cloud may have to share hardware, such as CPU, memory, etc., or software, like a hypervisor, with other cloud users. We assume that malicious users can launch various attacks, including but not limited to internal cloud attacks [17–20, 23, 24], side-channel attacks [22, 24]. Under these threats, components of a multi-cloud system running in the public environment can be compromised by any attackers. Since some components may have very high privilege, once these components are

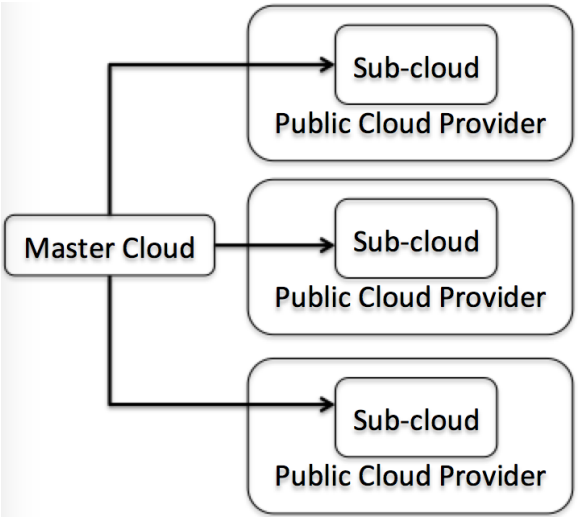

Figure 4.3: SECross Model: Master and Sub-cloud

compromised, attackers can potentially gain high privilege to attack the rest of the system.

4.3 SECross Model

We developed SECross model to enhance a multi-cloud system resilience to cyber-attacks. Our multi-cloud system provides infrastructure as a service (IaaS). SECross model has two types of cloud systems, master cloud and a set of sub-clouds as shown in Figure 4.3. Master cloud has a private environment, and its resources, such as CPU, memory, image files, network, etc., are not shared with public users. In our implementation, we deploy master cloud in the datacenter of NCSU which can only be accessed by authorized NCSU users. NCSU cloud users can directly use resources in the master cloud. However, when requests go beyond its capacity, SECross master cloud can deploy extra resources from any of its sub-clouds. We improve compromise resilience in two ways. First, since sub-clouds are deployed on public clouds, and they potentially have more possibility being compromised by various outside attacks, master cloud needs to resist compromises in any of sub-clouds. Second, if attackers are able to compromise master cloud either by using one compromised sub-cloud or attacking master cloud directly, uncompromised sub-clouds need to resist attacks from the compromised master cloud and sub-clouds.

We describe SECross model from several perspectives. First, we illustrate the architecture of SECross model and introduce how SECross works in a multi-cloud environment. Second, we describe the access rule among SECross components and the firewall rules to enforce that. Third, we show how SECross model handles users authentication and authorization. Fourth, we show how users securely log in and access remote reserved computing resources. Fifth we

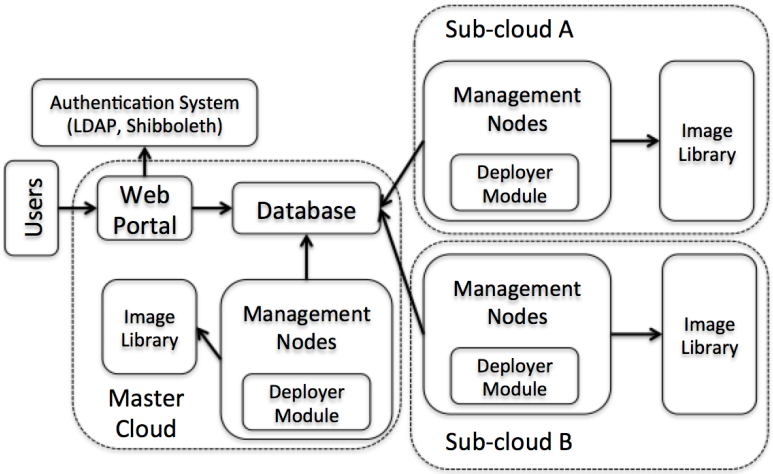

Figure 4.4: SECross Architecture

design a fine-grain access control policy for components accessing SECross database. Finally, we discuss how a SECross management node can monitor users' requests and reject the requests that appear out of order.

4.3.1 SECross Multi-cloud Architecture

SECross model has five major components, SECross web access portal, SECross database, SECross management node and authentication sub-system as shown in Figure 4.4. It has two types of cloud environments which are master cloud and sub-cloud. Master cloud is hosted in a private environment but has limited resource capacity. Sub-cloud is hosted in a public environment usually managed by independent organizations. Sub-cloud can run in any available IaaS public clouds with potentially extremely large resource capacity.

A SECross web portal is currently a graphical user interface (GUI) for all users. Users can log in to a VCL system, send requests, manage resources and configure privileges via the web. Web portal can communicate with a specific authentication system to verify users' identity. It also accesses SECross database and stores users' commands in SECross database. SECross web portal cannot access any other components of a SECross system.

A SECross database is the database of the multi-cloud system. It contains user information, user privileges, various resources information, such as computing resources and image resources, user service requests, SECross management node information, etc. Since some information recorded in SECross database is sensitive, we have to carefully define the fine-grain access control policy for a SECross web portal and SECross management nodes.

SECross model can have multiple management nodes, and each management node works

independently to manage computing resources in each sub-cloud or data center. In Figure 4.4, one management node manage computing resources in the master cloud, and the other two management nodes control the computing resources in sub-clouds A and B respectively. Management node in a master cloud could actually be an option. This depends on whether there are available computing resources in the master cloud. If computing resources are only existing in sub-clouds, SECross model can only host the web portal and the database in the master cloud.

A SECross image library stores images files. Each SECross management node has its own SECross image library which cannot be accessed by any other SECross components. A SECross management node can provision computing machines by directly using the image files from its SECross image library, and update the image or create a new image when users request.

The goal of SECross model is to design a compromise-resistant multi-cloud system. Since a management node can run in a third party cloud, we need to be careful when we design management node communications with the rest of multi-cloud system. Based on our assumptions in Section 4.2, attackers are able to compromise one of management node in a sub-cloud. SECross model has to set up fine-grain access privilege, so that compromised management nodes cannot attack the system components in the master cloud or any other sub-clouds. If some critical components in the master cloud are compromised, such as the SECross database or the SECross web portal, SECross model must have a mechanism for any un-compromised components in sub-clouds to detect malicious resource requests and even deny the requests if necessary. Because a management node controls all resources in a single cloud environment, we can use a management node to enforce a security policy in a sub-cloud to resist potential malicious requests.

4.3.2 Access Rules

SECross model defines the access rules for all SECross components. Only necessary network ports are opened to the target IP addresses in firewall to reduce the risk. As shown in the Figure 4.4, SECross web portal and SECross management nodes can access SECross database. Each SECross management node has and can only access its own SECross image library. SECross can synchronize image files among SECross image libraries in different cloud environments, but it should be only performed SECross administrators. All other accesses among SECross components are forbidden.

In the firewalls rules, all SECross components open an SSH login port for administrators to configure SECross system. The administrators should use public/private key for SSH authentication. The private key of a SECross administrator should be stored securely and separately from the SECross system. SECross management nodes should not be accessible in other ways (with the exception of a physical console). SECross web portal opens port 443 for HTTPs

access. SECross database allows accessing from port 3306 for mysql database queries. Since both SECross web portal and SECross management nodes usually have fixed IP addresses and are running for a long period, SECross only allows these fixed IP addresses to access the database.

4.3.3 User Authentication

In SECross model, the user authentication system is independent of the rest of multi-cloud system and can be only accessed in a controlled fashion. In the prototype, it is a web portal. The web portal has only the privilege to verify users' identity by his or her username and password and retrieve users' group information from the authentication system. An advantage of this design is that the password related information, such as the hash values of passwords and salts, is not stored in SECross database. Compromising of SECross system cannot leak the users' most sensitive information, or modify any users' identity information. Using an independent authentication system is also a cost-effective and efficient way to manage users in a large organization, such as universities or companies.

LDAP and Shibboleth are widely used protocols to authenticate users. SECross allows any organizations to maintain and manage their users information in independent LDAP or Shibboleth servers. SECross only uses both protocols to verify users' identities, such as users' names and users' group, without storing any sensitive users' information, such as passwords or the hash values of passwords. If SECross compromising is compromised in part, SECross can avoid leaking most of the information related to users.

When a user tries to log in to the system, the portal will send the username and password to the authentication system (e.g. LDAP server) through an encrypted network channel. After verifying user identity, the authentication system will send back the information, such as the user name and the group names the user belong to. If user is not already in the database, username and user's groups will be inserted into the database. Through the entire process, SECross only records user names and groups.

4.3.4 Computing Resources Access

When SECross system provisions a computing resource for a user, such as a virtual machine (VM) or a bare-metal machine. Using the same, single-sign-on password used to login SECross may be an issue if the resource, or the remote sub-cloud, has already been compromised. A one-time password can be used for every reservation in SECross, but SECross database has to store all passwords and usernames. If SECross database is compromised, all current reservations can be compromised no matter where these machines are running because attackers can know the target IP addresses, usernames, and passwords.

It is better to use private/public key authentication for users to login computing resources.

Every user creates a pair of private key and public key. Only public keys are stored in SECross database. SECross management node retrieves the proper public key from the SECross database to set it up in computing resource. Then the user can use the private key to log in to the computing machine via SSH.

SECross management nodes are required to log in to any computing machines since SECross management nodes have to perform configuration operations in computing machines for users, like add a user account, configure SSH server, set up firewall rules, etc. SECross management nodes use SSH private/public key to authenticate themselves to login computing machines. But each SECross management node uses a unique key pair. The public key of SECross management node is loaded and configured in the image files. When the computing machine is booted from the image, SECross management node can directly login the machine by using its private key. The images in a cloud can only store the public key of the SECross management node of the cloud. If one image is transferred from one cloud to another, the old public key needs to be deleted, and new public key from the new SECross management node must be added. The private key of SECross management node is only stored locally on the SECross management node. It is not accessible by any other SECross components except for itself.

Since SECross uses unique pairs of public and private keys to authenticate both users and SECross management nodes, attackers are less likely to compromise SECross resources except for computing resources managed by the compromised components themselves. For instances, a compromised SECross management node cannot access to any VMs managed by another SECross management node since the proper private key is not available, a compromised SECross database would not have enough information to login any VMs because it does not have user private keys or SECross management node's private keys. And a compromised SECross web portal cannot intercept or access any private keys from users or SECross management nodes.

4.3.5 Database Access

As shown in Figure 4.4, there are two types of SECross components access the SECross database: SECross web portal and SECross management nodes. In order to access the SECross database, all connections are required to be encrypted by Secure Sockets Layer (SSL) protocol. Every process also has an independent access control policy to access the database. The goal of the access control is preventing a compromised SECross management node from attacking SECross database or any other SECross management nodes.

Classification of SECross database tables

We classify SECross database tables into five categories: user tables, resource tables, privilege tables, request tables and log tables. We give the detail of the categories in following.

User tables contain information related to users, such as users' IDs, first names, last names, emails, users' public keys, users' preferences, etc. Since SECross uses an independent authentication system, users' passwords or their password hash values are not stored in the database. An ID is a unique identifier for each user, it cannot be changed once imported from the authentication system. A pair of an ID and a public key is used for a user to login computing resource.

Resource tables record two types of resources, including computer resources and image resources. The computer resource tables define properties of a computing environment, such as the name of a computer, computer type (e.g., VM or bare metal), IP addresses, mac addresses, provisioning engine, the platform, memory size, CPU speed, the number of cores, network speed, etc. The image tables contain the image name, description of usage, and minimum requirement for running environment, such as memory size, CPU speed, a number of cores, network speed, etc.

Privilege tables define users' groups, image groups, computers' groups, users' privileges and user groups' privileges. The user groups table specifies the groups a user belonged to. The image group table defines group membership for all images. The computer group table specifies in which computer groups a computing unit belongs to. User privilege tables and user group privilege tables determine which resources users can access to and what management operations users can perform. There are mainly six types of privilege in SECross, including computer resource management, image resource management, group management, SECross management node administration, resource grant and image reservation. Computer resource management or image resource management indicates whether a user or a user group has the privilege to change a content of a computer resource or image resource accordingly. Group management specifies if a user or a user group own the privilege to manage users' group, images' group and computers' group. SECross management node administration defines whether a user or a user group can change a setting for SECross management node. Resource granting specifies if a user or a user group can grant other users or user groups to access certain computer resources and image resources. Image reservation indicates whether a user or a user group can request a computing environment for the specific image or image group. A well-defined privilege tables can ensure that any legitimate users can only access and manage resources predefined.

When a user makes a request for a computing environment, a new request will be inserted into request tables. They define information related to this reservation of the computing environment, including the target computer resource and image resource, the user's id, the reservation time period, the reservation state, the SECross management node which is provisioning the computing environment, etc.

Log tables record varies information in SECross for the usage statistically analysis, system activities monitoring, etc. It stores information such as a user login, a service request, a

	User	Resource	Privilege	Request	Log
Web Portal	SIU	SIUD	SIUD	SIUD	SI
Management Node	S(Partial)	S(Partial)	N/A	SIUD(Partial)	I

Table 4.1: Database Access Control Policy. S, I, U, D represent which operation is allowed to execute on database tables or Columns. S means select, I is insert, U indicates update and D means delete.

configuration change, etc.

Database Access Policy

There are four basic database operations can be performed in SECross database, which are select, insert, update and delete. We assume that SECross web portal and SECross management node do not have privileges to perform other operations, such as grant, revoke, create table, delete table, etc. As shown in Table 4.1, we define the SECross database access privilege for SECross web portal and SECross management nodes.

SECross web portal is a user interface to interactive with users and running in the master cloud which is a private environment. It requires having a high privilege to access SECross database because common users and administrators rely on it to use and manage SECross system. It has all four operation privileges on resource tables, privilege tables, and request tables. Since SECross system can disable a user account by changing the user's status from active to inactive or removing the user from all available user groups, SECross web portal does not need delete privilege on user tables. Thus, SECross web portal can perform only select, insert and update operations on user tables. Because log tables are used to record all activities and analyze cloud usage statistics, SECross web portal only needs to insert and select operations on the SECross database.

Since SECross management nodes can run in a public cloud environment, which can potentially be attacked by internal cloud attacks, SECross model only gives the least privilege to them on accessing SECross database. Firstly SECross model uses **column level access control** on user tables and resource tables for SECross management nodes. While provisioning a computing environment, a SECross management node has to fetch the user's ID, public keys from user tables, and the computer configuration (e.g., memory size, CPU speed, etc.) and the image information (e.g., the image name, the image location, etc.) from resource tables. So SECross model uses column level access control to allow a SECross management node access only these columns.

When a user makes a reservation in SECross, it is SECross web portal to check whether the user has proper privilege to provision the request. If the user has the right privilege, SECross

web portal chooses which location the computing machine should be provisioned, and inserts this provisioning information into SECross request tables. If the user does not have the privilege, SECross web portal will deny the request. In this way, SECross management nodes do not need to access SECross privilege tables.

SECross management nodes insert provisioning information into SECross log tables but does not require to read or modify the log tables. Therefore, SECross model only allows SECross management nodes to perform an insert operation on the log tables.

SECross request tables store all information related to users' service requests, SECross management nodes have to perform select, insert, update and delete operations in order to provision computing machines. However, a compromised SECross management node can easily leak or modify users' reservations which are managed by itself or other legitimate SECross management nodes. For instances, an adversary can delete a user's reservation, or change the image chosen by a user and enforce she or he to use a compromised image. Therefore, SECross model uses **row level access control** on SECross request tables to monitor SECross management nodes' access. Every SECross management node can only select, update, delete a row on SECross request tables when the row belongs to itself. Since MySQL does not support row-level access control, we use database trigger to enforce this access control. We introduce more detail of the implementation in Section 4.5.

4.3.6 SECross Management Node Monitor

SECross model resists potential attacks from SECross web portal and SECross database. In SECross model, a compromised SECross web portal or SECross database cannot access SECross management node directly as described in Section 4.3.2, or login any computing machines without compromising computing machines' SECross management nodes as discussed in Section 4.3.4. But a compromised SECross web portal or SECross database can send malicious requests to any legitimate SECross management nodes by modifying SECross request tables. A SECross management node has to monitor the request from SECross database, and alert SECross administrators or even deny the requests when the requests look like suspicious.

First, a SECross management node records the requests it provisions in a local log file. This includes users' ID, reservation time, etc. The log file can be used by SECross administrators to trace and audit any unusual requests. Second, a SECross management node has maximum resources request limitation. A SECross management node can monitor current computing resources usage in its management zone. When the maximum number of computing machines are deployed by the SECross management node, it will reject other requests until some of the current reservations end and the computing resources become available. Third, SECross can define a SECross request policy for SECross management node to decide whether a computing

resource request is legitimate. The policy defines the restrictions of computing machines that a SECross management node can deploy in its domain. The restrictions include maximum CPU speed, maximum memory size, maximum network speed, blocked images, etc. Since the SECross management node's policy file is stored locally on the SECross management node machine, so it cannot be accessed by any other SECross components.

4.4 Security Analysis

We discuss the compromise resilience of SECross model from two major perspectives. Since some components of SECross model are running in a public cloud environment, first we analyze the security level of SECross model when the components in a public cloud environment are compromised. Second, we assess the security level of SECross model even when the components in the master cloud are attacked.

4.4.1 Public Cloud Adversaries

Some of SECross management nodes and computing machines are running in a public cloud environment. Sharing of resources with public cloud users occurs. An attacker can impersonate a normal public cloud user to launch internal cloud attacks on SECross computing machines or a SECross management node in the same cloud.

If a computing machine is compromised, an attacker can access the resources in the computing machine including the user's public key, the SECross management node's public key and the data on the machine. But the attacker cannot access any other SECross components.

When an attacker can compromise the SECross management node, the adversary can access the resources that belong to this SECross management node including the SECross management node's image library, computing machines, and access the certain tables, columns and rows in SECross database as discussed in Section 4.3.5. However, the compromised SECross management node cannot access others' resources not managed by itself, such as other SECross management nodes' image libraries, computing machines or other SECross management nodes. The compromised SECross management node can see some information on users and resources from SECross database, like users' IDs and public keys, and computing resources configurations and images' configurations. But the compromised SECross management node cannot update, delete or insert information on SECross user tables and resources tables. Since SECross model has the row-level access control on SECross request tables, the compromised SECross management node cannot observe, modify requests of other SECross management nodes, or insert new requests to other SECross management nodes.

When attacks are launched in a public cloud domain, SECross model can resist the spread of

the compromise from one cloud to another cloud. SECross multi-cloud model exhibits compromise resilience to internal cloud attacks in a public cloud.

4.4.2 Other Adversaries

SECross web portal and the database are running in a private environment, primarily to avoid internal cloud attacks. However, attackers can use traditional methods to attack a SECross multi-cloud system. SECross web portal is the only user interface which can be accessed by public users, therefore, it becomes a popular target for attackers to launch a variety of attacks such as SQL injection, buffer overflow, etc. If a SECross web portal is compromised, an attacker can have a high privilege to access SECross database according to Section 4.3.5. But the attacker cannot access the existing computing machines in other sub-clouds because attacker will not be able to access to proper user private keys. The attacker cannot access any of SECross management nodes and image libraries. However, the attacker can impersonate a legitimate user to send malicious reservation requests to any SECross management nodes for new computing machines. The SECross database and SECross management nodes can log the attacker's activities for the intrusion detection analysis, and SECross management node can deny the service request directly if the request violates its service policy as stated in Section 4.3.6.

If SECross database is compromised, the adversary cannot access any legitimate SECross web portal, SECross management nodes, image libraries or existing computing machines. Similar to a compromised SECross web portal, the adversary can send malicious requests to any SECross management nodes, but SECross management nodes are able to log the requests and even deny them. The adversary has the ability to hide some of the traces by deleting content on SECross log tables. However, any mismatch between SECross database log tables' content and SECross management log content can indicate potential compromising is happening.

4.5 Implementation

We built a multi-cloud system by using apache open source cloud system VCL 2.4.2. SECross web portal and SECross database were running on two VMs on the desktop machine in our lab which is running Windows 10 64 bits and VMware Workstation 12. Our desktop machine was running in our university network to simulate master cloud environment which has a private hypervisor and hardware. We used IBM Softlayer to reserve a VM with CentOS 6.5 64 bits to run a SECross management node. We developed a Softlayer deployer module to deploy computing machines in IBM Softlayer by using IBM Softlayer API, so VCL is able to deploy computing machines in not only traditional computing environment, such as university's host machines which are running KVM or VMware ESXi as a hypervisor, but also industry public clouds.

We used the university's LDAP server as SECross's authentication system. A SECross web portal uses PHP to communicate with the SECross database, and a SECross management node uses Perl to operate and communicate with SECross database and IBM Softlayer management system.

We used MySQL 5.1.73 as our SECross database. MySQL supports user-level, table-level, and column level access control but lacks of row-level access control. In order to simplify our implementation, we use view table and trigger operation of MySQL to enforce row-level access control. Every SECross management node has a unique ID, and the ID is used as MySQL account name for the SECross management node. On SECross request tables there is one column which stores the SECross management node ID, named managementID. SECross model does not allow SECross management nodes access this column. We create a trigger for insert operation on SECross request tables with the operation "SET NEW.managementID = substring_index(user(), '@', 1)". Whenever a SECross management node inserts a request the managementID can record the ID of the SECross management node. We use MySQL view table for the select operations performed by SECross management nodes with condition "managementID = substring_index(user(), '@', 1);". Therefore, a SECross management node can only observe request information that belongs to itself. We create a trigger for update and delete operations on SECross request tables with condition "IF managementID <> substring_-index(user(), '@', 1) THEN SIGNAL SQLSTATE '02000' SET MESSAGE_TEXT = 'Error'; END IF;" Therefore when the user ID does match the SECross management node ID, the update or delete operation will fail.

Chapter 5

Related Work

5.1 Hadoop Security Related Work

The overloaded authentication keys vulnerability was discussed in the Hadoop security design report [9], but the authors of that report were relatively dismissive of its impact. We believe that a public cloud may exacerbate this and other vulnerabilities that may stem from the fact that the application was originally not designed for use in a shared cloud-based environment. A number of researchers have investigated at Hadoop security in the cloud [34, 64–70]. Some offered security improvements directed at Hadoop [71–73], and some improve Hadoop security by improving the security of overall cloud environment [1, 2, 21, 74].

Hadoop network connection protection. Several projects provide secure channels for Hadoop's various communications. Myers provides an encrypted channel for data transfers between a client and a Data Node [71]. Abdelnur uses HTTPS to replace HTTP to send command and transfer files in Hadoop [72, 73]. These projects ensure that all network communications within Hadoop occur through encrypted channels. This helps prevent eavesdropping and man-in-middle attacks, but Hadoop's security still relies on the integrity of all its components.

Data protection in cloud. Not trusting disks and networks, Tahoe [34, 75] stores encrypted data in a distributed file system. Unlike HDFS, all data in Tahoe are encrypted and integrity checked by the gateway. File system server cannot read or modify the data. The project Hadoop-lafs [64] provides an interface to Tahoe. An Hadoop Map-Reduce job can use the data in Tahoe distributed file system. This approach provides better protection for the data in a cloud, however, there is not fine-grained access control to manage map and reduce processes access to data.

Access control in Hadoop. Hadoop uses a POSIX-like model of access control [66]. It has an access control list for various Hadoop services. Every file and directory in HDFS have their own permissions for different kinds of users. Orchestrator [65] works on Hadoop and provides role-based access control (RBAC). These access control mechanisms rely on the integrity of

Hadoop.

Hypervisor security. Virtualization plays an important role in clouds. Hypervisor generally is the highest privilege management system to enforce security. Hyper Sentry [21] uses techniques to ensure a hypervisor's integrity. HIMA [1] is a hypervisor-based agent to measure the integrity of user programs in VM. These methods can improve the overall security of a Cloud environment.

5.2 Cloud Storage Security Related Work

Many use convergent encryption to encrypt and deduplicate data [25, 26, 33–35, 53, 57]. However using only convergent encryption is vulnerable to brute-force dictionary attack. Approach called Dupless uses an independent key generation server to resist such attacks [25]. Dupless basically forces attacks to be real-time instead of off-line. Real-time attacks can be recognized, slowed down, and countered. Yitao introduces a new distributed key generation protocol to use a group of users' clients to generate encryption keys [57]. Jian et al. use password-authenticated key exchange (PAKE) protocol to obtain the encryption key from the previous user who has uploaded the identical file [76]. The user's client can protect itself from online brute-force attack by limiting the number of PAKE instances they will participate in for each file. And Liu shows that, when attackers can compromise multiple clients, this approach is more resistant to online brute-force attacks. However, Liu's method requires that certain number of clients need to be online, so these online clients can communicate with each other. Frederik et al. use server aid method, which is similar to Dupless, to generate an encryption key, and thus enable a cloud provider to attest to its clients the deduplication patterns of stored data [77]. But again, the system only slows down the online brute-force dictionary attack.

Compared with client-side encryption methods in [25, 57, 76], SafeDedup uses Cloud Storage server to process data key generation and data encryption. Clients encrypt data using symmetric encryption when they send data to the servers in both Dupless and SafeDedup. Therefore, Dupless and SafeDedup should have similar performance on the client side. Dupless uses the same key to encrypt the data if the data content is the same, but SafeDedup uses a random key to encrypt data in every write request. If attackers are a CSS internal adversaries and are able to eavesdrop on the victim's network, they can intercept the encrypted data and launch brute-force attack on Dupless but not on SafeDedup. SafeDedup can leverage the strong isolation between CCS VMs and host machine enforced by a hypervisor to provide a simple method to share data among users. This feature cannot be securely provided by existing client side methods. SafeDedup requires computing for data encryption on the server side compared with Dupless encrypts data on client side.

5.3 Multi-cloud System Security Related Work

Some researchers and companies have been developing multi-cloud infrastructure [43–47, 78–80]. It would appear that most of the security efforts are focused on data and work flows security running on the multi-cloud system [78–81]. Jensen et al. propose a security mechanism that can distribute work flow to multiple clouds [81]. Then an attack can be detected by comparing the same work flow on different clouds. Bessani et al. propose a storage system for data availability and confidentiality on multi-clouds [78]. Bowers et al. introduce HAIL, a high data availability and integrity layer for multi-clouds [79]. Massonnet et al. take security into account when select cloud providers [82]. Their Idea multi-cloud system decides where to deploy resources based on security constraints. From a different perspective, Patel et al. propose an intrusion detection and prevention system for a cloud system to enhance security of the cloud system [83]. Our SECross also focuses on the security of the multi-cloud system. But SECross resists attacks by enforcing isolation and fine-grain access control among SECross components .

Chapter 6

Conclusion and Future Work

In this dissertation, we report on our investigation of some of the issues when applications move into a cloud. We find that application and data isolation, access control, and efficient encryption can solve many of the issues. We use two specific case-studies - Hadoop application, and deduplication of encrypted data to highlight both the issues and some possible solutions. We also discuss secure use of multiple public clouds as a way of managing cloud provided computational or storage resources.

We studied the potential attacks on Hadoop in a public cloud. We designed and implemented SEHadoop model that consists of SEHadoop runtime model, SEHadoop Block Token, and SEHadoop Delegation Token to improve compromise resilience of "standard" Hadoop in a public cloud. SEHadoop model enhances isolation level among Hadoop components and enforces least access privilege on Hadoop processes. Our experiments suggest that solution overhead is very low. We also discuss how to easily migrate Hadoop jobs to SEHadoop.

We also developed and implemented a way of deduplicating encrypted data that may be stored in a cloud-based storage system. SafeDedup leverages virtualization technology to isolate data from the cloud storage service (CSS) systems and increase CSS resistance to attacks. Fine granularity access control is used to enforce user-specific restrictions. SafeDedup's storage saving level is similar to convergent encryption. We discuss security impacts of various threats from both external and internal adversaries, and how SafeDedup resists these attacks, especially from malicious CSS inside users.

In the last part of the dissertation, we discuss the emerging demand for a multi-cloud system, and the potential new threats when some of the clouds are public. We developed and implemented SECross model for a multi-cloud system. SECross model defines secure access rules among SECross components, and describes how a SECross management node enforces its security policy. We discuss the security impacts when some components of SECross are compromised and discussed how SECross model resists attacks.

Based on the insights in this dissertation, we propose three research directions as future work.

Hadoop Security in a public cloud Hadoop job should leverage SEHadoop delegation token and SEHadoop block token to exploit fully the advantage of fine-grain access control enforced by them. We plan to study current popular Hadoop jobs and find a proper method to migrate Hadoop job to SEHadoop job. Based on SEHadoop delegation token and SEHadoop block token, a detection system can be built by monitoring whether there is an access request which violates the access control enforced by both SEHadoop tokens. The violation activities should be alerted and logged for administrators to analyze.

Cloud Storage Service Security in a public cloud We plan to modify the small part of HDFS code. Therefore, only one TCP connection is needed for one read operation. In this way, we can improve the read performance for SafeDedup especially for reading small files. In order to achieve better effectiveness to resist side-channel attack, we plan to introduce a random delay in the case of immediately reading after writing as discussed in Section 3.4.2. We plan to provide more options for users to read or write a file. While reading data from SafeDedup, users can choose to decrypt data by themselves or by a hypervisor. During writing data to SafeDedup, users can also choose to encrypt data by themselves or by a hypervisor. In this way, for extremely sensitive data, users can enforce client-side encryption and decryption.

Multi-cloud System Security in a public cloud We plan to build an extra access layer between SECross database and all other components. Only this layer can access SECross database and provide a set of API for all other SECross components to access SECross database. In this way, we plan to design more complicated and secure access control policies and more powerful monitors than the original architecture. We plan to develop an active SECross monitor system by fetching information from SECross database, including log tables, request tables, etc. We plan to use SECross management node monitors to detect suspicious behaviors and alert SECross administrators.

REFERENCES

[1] A.M. Azab, Peng Ning, E.C. Sezer, and Xiaolan Zhang. HIMA: A Hypervisor-Based Integrity Measurement Agent. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pages 461–470, 2009.

[2] Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. SICE: a hardware-level strongly isolated computing environment for x86 multi-core platforms. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 375–388, New York, NY, USA, 2011. ACM.

[3] Yinqian Zhang and Michael K Reiter. Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 827–838. ACM, 2013.

[4] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28(3):583 – 592, 2012.

[5] Donghoon Kim and Mladen A. Vouk. A survey of common security vulnerabilities and corresponding countermeasures for saas. In *Globecom Workshops, the Second International Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA), IEEE, 2014*, pages 59–63, Dec 2014.

[6] Donghoon Kim, H.E. Schaffer, and Mladen A. Vouk. Paas security countermeasures: A survey. In *3rd International 2015 IBM Cloud Academy Conference (ICACON 2015), Budapest, Hungary*, 2015.

[7] Hadoop. https://hadoop.apache.org/releases.html. Accessed in December 2015.

[8] Jimmy Wong. Which Big Data Company has the World's Biggest Hadoop Cluster. http://www.hadoopwizard.com/which-big-data-company-has-the-worlds-biggest-hadoop-cluster/. Accessed in November 2013.

[9] Owen O'Malley, Kan Zhang, Sanjay Radia, Ram Marti, and Christopher Harrell. Hadoop security design. *Yahoo, Inc., Tech. Rep*, 2009.

[10] Joyent. Joyent Solution for Hadoop. http://www.joyent.com/products/computeservice/features/hadoop/. Accessed in November 2013.

[11] Inforchimps. Cloud::hadoop. http://www.infochimps.com/infochimps-cloud/cloud-services/cloud-hadoop/. Accessed in November 2013.

[12] Amazon Elastic MapReduceg. http://aws.amazon.com/elasticmapreduce. Accessed in November 2013.

[13] A. Mandal, Yufeng Xin, I. Baldine, P. Ruth, C. Heerman, J. Chase, V. Orlikowski, and A. Yumerefendi. Provisioning and evaluating multi-domain networked clouds for hadoop-based applications. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 690–697, 2011.

[14] Hong Mao, Zhenzhong Zhang, Bin Zhao, Limin Xiao, and Li Ruan. Towards Deploying Elastic Hadoop in the Cloud. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, pages 476–482, 2011.

[15] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1 – 11, 2011.

[16] Cloud Security Alliance. Top Threats to Cloud Computing. https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf Accessed in April 2014.

[17] Kostya Kortchinsky. Cloudburst: A vmware guest to host escape story. http://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-PAPER.pdf. Accessed in November 2013.

[18] J. Rutkowska R. Wojtczuk. Xen 0wning trilogy. In Black Hat conference, USA, Aug. 2008.

[19] Secunia. Vulnerability report: Vmware esx server 3.x. http://secunia.com/advisories/product/10757/. Accessed in November 2013.

[20] Secunia Advisory. Xen PV Kernel Decompression Multiple Vulnerabilities. http://secunia.com/advisories/44502/. Accessed in November 2013.

[21] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang, and Nathan C. Skalsky. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 38–49, New York, NY, USA, 2010. ACM.

[22] Francisco Rocha and Miguel Correia. Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 129–134. IEEE, 2011.

[23] Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, Ahmad-Reza Sadeghi, and Thomas Schneider. AmazonIA: when elasticity snaps back. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 389–400. ACM, 2011.

[24] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[25] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. DupLESS: Server-aided Encryption for Deduplicated Storage. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 179–194, Berkeley, CA, USA, 2013. USENIX Association.

[26] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology–EUROCRYPT 2013*, pages 296–312. Springer, 2013.

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

[28] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 261–270, New York, NY, USA, 2010. ACM.

[29] A. Binbusayyis and Ning Zhang. Decentralized attribute-based encryption scheme with scalable revocation for sharing data in public cloud servers. In *Cloud Technologies and Applications (CloudTech), 2015 International Conference on*, pages 1–8, June 2015.

[30] Yang Lu and Jiguo Li. A pairing-free certificate-based proxy re-encryption scheme for secure data sharing in public clouds. *Future Generation Computer Systems*, 2015.

[31] Mrs N Manju and Mrs V Shanmugapriya. Data integrity and security in cloud computing using cryptography mechanism. *International Journal of Application or Innovation in Engineering & Management (IJAIEM), Volume 3, Issue 3*, March 2014.

[32] John R Douceur, Atul Adya, William J Bolosky, P Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624. IEEE, 2002.

[33] Paul Anderson and Le Zhang. Fast and secure laptop backups with encrypted de-duplication. In *LISA*, 2010.

[34] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, StorageSS '08, pages 21–26, New York, NY, USA, 2008. ACM.

[35] Freenet. https://freenetproject.org/. Accessed in March 2015.

[36] Peter Mell and Tim Grance. The NIST definition of cloud computing. In *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg*, 2011.

[37] IBM Softlayer. http://www.softlayer.com. Accessed in August 2015.

[38] Amazon Elastic Compute Cloud (Amazon EC2). https://aws.amazon.com/ec2/. Accessed in August 2015.

[39] Microsoft Cloud. http://www.microsoft.com/en-us/server-cloud/, Accessed in August 2015.

[40] Google Cloud Platform. https://cloud.google.com. Accessed in August 2015.

[41] M.A. Vouk, A. Rindos, S.F. Averitt, J. Bass, M. Bugaev, A. Kurth, A. Peeler, H.E. Schaffer, E.D. Sills, S. Stein, J. Thompson, and M. Valenzisi. Using vcl technology to implement distributed reconfigurable data centers and computational services for educational institutions. *IBM Journal of Research and Development*, 53(4):2:1–2:18, 2009.

[42] NCSU Virtual Computing Lab (VCL). https://vcl.ncsu.edu/. Accessed in August 2015.

[43] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier. A federated multi-cloud paas infrastructure. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 392–399, June 2012.

[44] Jiang Dejun, Guillaume Pierre, and Chi-Hung Chi. Resource provisioning of web applications in heterogeneous clouds. In *Proceedings of the 2nd USENIX conference on Web application development*, pages 5–5. USENIX Association, 2011.

[45] Gunho Lee, Byung-Gon Chun, and Randy H Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. *Proceedings of HotCloud, 3rd USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–5, 2011.

[46] Ralph Mietzner, Frank Leymann, and Mike P Papazoglou. Defining composite configurable saas application packages using sca, variability descriptors and multi-tenancy patterns. In

Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on, pages 156–161. IEEE, 2008.

[47] Navid Pustchi, Ram Krishnan, and Ravi Sandhu. Authorization federation in iaas multi cloud. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, SCC '15, pages 63–71, New York, NY, USA, 2015. ACM.

[48] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[49] Hadoop website. http://hadoop.apache.org Accessed in December 2013.

[50] Navid Nathoo. Cloud wars - the fall of cloud storage. cloudtimes. 2013, Accessed in July, 2014.

[51] Dropbox. https://www.dropbox.com. Accessed in December 2015.

[52] Mozy. https://mozy.com. Accessed in December 2015.

[53] Mark W Storer, Kevin Greenan, Darrell DE Long, and Ethan L Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 1–10. ACM, 2008.

[54] William J Bolosky, Scott Corbin, David Goebel, and John R Douceur. Single instance storage in Windows 2000. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 13–24. Seattle, WA, 2000.

[55] Atul Adya, William J Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R Douceur, Jon Howell, Jacob R Lorch, Marvin Theimer, and Roger P Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

[56] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *FAST*, volume 2, pages 89–101, 2002.

[57] Yitao Duan. Distributed key generation for encrypted deduplication: Achieving the strongest privacy. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, CCSW '14, pages 57–68, New York, NY, USA, 2014. ACM.

[58] AMD V. http://www.amd.com/en-us/solutions/servers/virtualization. Accessed in May 2015.

[59] Intel VT. http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html. Accessed in December 2015.

[60] Z Wilcox-O'Hearn. Convergent encryption reconsidered, 2011.

[61] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.

[62] QEMU, http://wiki.qemu.org/. Accessed in December 2015. Accessed in July, 2014.

[63] KVM, http://wiki.qemu.org/kvm. Accessed in December 2015. Accessed in July, 2014.

[64] Hadoop-lafs project. https://code.google.com/p/hadoop-lafs. Accessed in November 2013.

[65] Zettaset Orchestrator. http://www.zettaset.com/. Accessed in November 2013.

[66] Inc. Andrew Becherer, iSEC Partners. Hadoop security design just add kereros? really?

[67] Jason Cohen and Subatra Acharya. Towards a more secure apache hadoop hdfs infrastructure. In *Network and System Security*, pages 735–741. Springer, 2013.

[68] Jiaqi Zhao, Lizhe Wang, Jie Tao, Jinjun Chen, Weiye Sun, Rajiv Ranjan, Joanna Kołodziej, Achim Streit, and Dimitrios Georgakopoulos. A security framework in g-hadoop for big data computing across distributed cloud data centres. *Journal of Computer and System Sciences*, 80(5):994–1007, 2014.

[69] Devaraj Das, Owen OMalley, Sanjay Radia, and Kan Zhang. Adding security to apache hadoop. *Hortonworks, IBM*, 2011.

[70] Lifei Wei, Haojin Zhu, Zhenfu Cao, Xiaolei Dong, Weiwei Jia, Yunlu Chen, and Athanasios V Vasilakos. Security and privacy for storage and computation in cloud computing. *Information Sciences*, 258:371–386, 2014.

[71] Aaron T. Myers. Add support for encrypting the datatransferprotocol. https://issues.apache.org/jira/browse/HDFS-3637. Accessed in November 2013.

[72] Alejandro Abdelnur. Add support for encrypted shuffle. https://issues.apache.org/jira/browse/MAPREDUCE-4417. Accessed in November 2013.

[73] Alejandro Abdelnur. Add support for HTTPS to the web UIs. https://issues.apache.org/jira/browse/HADOOP-8581. Accessed in November 2013.

[74] Jicheng Shi, Xiang Song, Haibo Chen, and Binyu Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 194–199. IEEE, 2011.

[75] Tahoe-LAFS website. https://tahoe-lafs.org/trac/tahoe-lafs/wiki. Accessed in November 2013.

[76] Jian Liu, N. Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 874–885, New York, NY, USA, 2015. ACM.

[77] Frederik Armknecht, Jens-Matthias Bohli, Ghassan O Karame, and Franck Youssef. Transparent data deduplication in the cloud. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 886–900. ACM, 2015.

[78] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12, 2013.

[79] Kevin D Bowers, Ari Juels, and Alina Oprea. Hail: a high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM, 2009.

[80] Christian Cachin, Robert Haas, and Marko Vukolic. Dependable storage in the intercloud. Technical report, Research Report RZ, 2010.

[81] Meiko Jensen, Jörg Schwenk, Jens-Matthias Bohli, Nils Gruschka, and Luigi Lo Iacono. Security prospects through cloud computing by adopting multiple clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 565–572. IEEE, 2011.

[82] Philippe Massonet, Jesus Luna, Alain Pannetrat, and Ruben Trapero. *Engineering Secure Software and Systems: 7th International Symposium, ESSoS 2015, Milan, Italy, March 4-6, 2015. Proceedings*, chapter Idea: Optimising Multi-Cloud Deployments with Security Controls as Constraints, pages 102–110. Springer International Publishing, Cham, 2015.

[83] Ahmed Patel, Mona Taghavi, Kaveh Bakhtiyari, and Joaquim Celestino Júnior. An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications*, 36(1):25–41, 2013.