Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3242

Data security in multi-tenant environments in the cloud

Tim Waizenegger

Course of Study:	Computer Science
Examiner:	Prof. Dr. Bernhard Mitschang
Supervisor:	Dipl.-Inf. Thomas Ritter
Commenced:	1. October, 2011
Completed:	13. April, 2012
CR-Classification:	C.2.4, C.5.5, D.2.11, H.3.4, K.6.5

Contents

List of Figures

List of Tables

Abstract

While cloud computing is widely used in consumer applications, business and enterprise customers remain hesitant. The most commonly cited issues preventing the adoption of cloud computing are reliability, security and privacy. [SKS11]

Enterprise Software as a Service solutions offered in the cloud consist of many distinct components that are integrated into a solution which is consumed by the customer. Single components are connected and form a complex solution by communicating and complementing their services. This communication is often not properly secured because components were developed for non-cloud scenarios where inter process and component communication security requirements are less stringent. Preventing unauthorized access by users, processes or components is a basic requirement for any solution. Especially in a cloud context the integration of not or lesser trusted components might be required but a trustable solution is still expected.

As a first line of defense, access to systems and services is secured by authentication mechanisms. This requires a system to validate user credentials as well as provide proof of its identity to the user. The individual components comprising a cloud service need to authenticate each other as well in order to prevent unauthorized access by compromised components or systems. Securing this communication by authentication requires the individual components to have access to certain keys. While authentication is used to secure services against unauthorized access, encryption can often be employed to secure data for transport or storage. In both cases similar problems are faced. When using keys for encryption and authentication the security of the system relies on securely managing the keys.

This thesis will investigate technology options for authentication, encryption and key management in a cloud based Software as a Service solution exemplified by the IBM SmartCloud Archive.

1 Introduction

Thesis Outline

This thesis will discuss the topic of security in a cloud computing context by focusing on the aspects of data security through encryption and secure system operation through authentication. Both of these concepts will be shown to rely on the secure management of keys and credentials, introducing the topic of key management systems. The presented solutions are motivated by addressing the following problems:

- What solutions exist for encrypting data for storage and transport and how can they be used in a cloud computing context?
- What mechanisms for authenticating users and systems are available today and what are their implications for key management?
- How can a secure key management solution be deployed in a cloud computing environment?
- Who generates the keys and credentials, the customer or the cloud provider and how can a key infrastructure like PKI be used in this context?
- Which keys does the cloud software require access to in order to perform certain tasks. Can they be derived from user credentials in order to prevent storing them in the cloud?
- How are keys and credentials stored in the cloud, what solutions exist and what are their shortcomings?

This thesis is divided into three parts leading from an introduction into the matter and technologies to an evaluation of the IBM SmartCloud Archive product to arrive at a recommendation and design for a key management system.

Chapter 1 will provide an overview of the topic of cloud computing, security in general and the IBM SmartCloud Archive.

Chapters 2 through 4 give insight into technologies and procedures that are essential for the security of a system.

Chapter 5 presents common security issues and guidance for securing a system against the underlying vulnerabilities and then analyzes the components that make up IBM SmartCloud Archive to derive security recommendations and requirements for a key management solution.

Chapter 6 foots on the results and possibilities provided in the previous chapters and presents the design for a secure key management solution.

Chapter 7 sums up the results of this thesis and provides reference points for further research.

1.1 Cloud Computing

The paradigm of cloud computing follows in the footsteps of cluster and grid computing. While a cluster is usually comprised of machines with a similar hardware configuration residing within the same location and network, a grid can be widespread and heterogeneous in its hardware configuration. Both of these models have in common that they are used for a single, or a very limited number of purposes like a specific scientific application, or the provision of a network or Internet service. They usually only serve a single organization, or a specific group of customers.

Cloud computing goes one step further than these models and creates a heterogeneous, widespread platform for multipurpose computing, available to a large user base. With this premise it becomes attractive for a wide range of computing applications that did not fit the requirements for cluster or grid computing.

Cloud computing seeks to get a foothold in markets formerly dominated by conventional IT systems by offering attractive advantages to its customers. The main cause for enterprises to employ cloud computing is the reduction of costs in all sectors associated with IT infrastructure while benefiting from higher quality services and software. Customers no longer have to maintain and implement their own infrastructure, but acquire these services from external vendors allowing them to focus on their core business. The pay-as-you-go model of cloud computing allows customers to shift their expenses from the capital (CAPEX) to the operational (OPEX) side, minimizing up-front investment and improving flexibility in times of lower demand.

The paradigm shift introduced by cloud computing to outsource data to third parties makes enterprises hesitant to move towards the cloud. The fear of losing control over their data which could lead to data loss, leaking or provider lock-in is a hurdle that has to be overcome by cloud computing before finding mainstream adoption among enterprise customers. [CGJ+09]

From an organizational point of view three roles involved in the cloud computing model can be identified. The *cloud infrastructure provider* operates and maintains the required hardware while the *cloud service provider* creates and services the software used by the *cloud customer*.

In order to achieve its goals like cutting costs, raising availability and security and providing a better service, cloud computing allows for the flexible delegation of these roles to different entities. This creates the sub models *public and private cloud* as described later in this chapter.

Through this delegation of roles it is possible for different groups of customers to take advantage of different aspects of cloud computing. While a large corporation can afford to operate its own cloud infrastructure in order to benefit from the higher efficiency the technology offers, smaller customers can leave this role to a cloud provider and benefit from higher quality software and better service than they could provide by themselves. [IT10]

Cloud Technologies

Cloud computing on itself is not a new technology but rather a means of combining proven, existing technologies to deliver a new kind of service. The following sections give a brief overview over the technologies that make up today's cloud computing infrastructures.

The technologies running cloud computing can be assigned to the same three roles identified before. The cloud infrastructure provider separates the offered service from the hardware it is running on by means of virtualization as described in the next section. This allows a flexible choice of hardware and enables the cloud service provider to create *elastic* services that scale up and down by starting and stopping virtualized instances. The technologies used by the cloud service provider to create such services are no different than in conventional web services and include distributed file systems, databases and application servers that comprise the middleware layer.

Most cloud services are delivered to the customer through rich web interfaces created with *HTML5* and *AJAX* technologies. Though it is possible to use native applications in combination with cloud services, this approach is most commonly used on mobile devices today. [Pha10]

this yields a cloud computing software stack that contains off-the-shelve components on the infrastructure and middleware layer which are only customized through configuration and set up, and custom components comprising the service and user interface layer that form the core of the cloud application.

Virtualization

Virtualization allows an operating system to run not directly on the physical hardware but rather on virtualized hardware. This layer of virtualization makes it possible to offer multiple instances of virtualized hardware on a single physical machine, each running a separate operating system instance.

The concept of virtualization first appeared in 1966 in IBMs System/360 mainframe computer. Development has continued and today virtualization is available in one form or another through nearly all available hardware platforms and operating systems. [AAB+05]

The main component of virtualization is the Hypervisor. It interfaces with the physical hardware and provides the virtual hardware instances used by its guest operating systems. Depending on the support for virtualization available in the hardware and operating system,

different types of Hypervisors such as hardware, software or paravirtualization are used. Modern hardware featuring virtualization support allows the Hypervisor to directly offer access to the physical hardware to its guest, while the Hypervisor stays in control over which resources exactly are accessible by the guest. This *hardware virtualization* has virtually no performance losses compared to native execution, making it an attractive solution for production environments.

When there is no support for virtualization in the underlying hardware, every hardware access by the guest needs to be passed through the Hypervisor to allow enforcing access limitations. On the software side, operating system support is not necessarily required for efficient virtualization if hardware virtualization is available, since the operating system has direct access to hardware as it would in a native environment. If no hardware virtualization is available though, the guest operating system can be harnessed to still achieve high performance by integrating parts of the Hypervisor directly into the kernel. This way the Hypervisor does not have to analyze and modify each hardware access since it directly controls the *paravirtualized* operating system and its hardware access.

Cloud Models

Public cloud computing is the concept that shapes today's Internet. It is a cloud service that is openly accessible from the Internet, though it might require authentication to gain access. Such an open environment poses certain security risks which makes this model unpopular with businesses but due to its convenience the preferred one for consumer services.

For customers with high security demands another model, the *private cloud* is used. Such a cloud can be hosted within the customer's premises, offering the highest security standard possible, or in an external data center managed by a third party. In both cases the cloud is only accessible from within the customer's local network, usually via VPN in case of an external cloud. Certain components of a cloud solution might have lower security requirements than others, yielding the *hybrid cloud* model. Critical components are housed in a private cloud while public data or services can be moved to a public cloud in order to leverage the lower cost of public cloud environments. [PM11]

Cloud Service Models

A cloud computing infrastructure can be used to deliver different types of services which are located on a specific layer of the Software-Platform-Infrastructure (SPI) model. From the bottom up these layers comprise the following service models. [PM11]

Infrastructure as a Service (IaaS) offers the lowest level of service access in cloud computing by providing the customer with access to computing resources like processing, storage and network. Often the customer can deploy an operating system of his choice or use a preconfigured operating system instance with full administrative access. Customers usually

access the service via a web interface or API which enables them to acquire virtual machines, upload, run and stop instances of VM-images and configure storage and network resources. The customer himself has to take care of scaling his application by provisioning additional virtual machines, and routing traffic to them in times of peak load. Through the use of the provided API he can integrate this functionality into his application and harness the elasticity of the cloud. The largest IaaS providers today include *Amazon EC2 and S3*, *Rackspace*, *Dropbox* and the upcoming *IBM SmartCloud Enterprise*.

Platform as a Service (PaaS) offerings can be built on top of Infrastructure as a Service or provided as an independent offering. The *platform* is a framework for running applications in the cloud. It usually consists of a runtime environment for a programming language like Java, C# or Python and a set of APIs allowing the developer to gain access to specific services offered by the cloud provider.

These services include access to a database, file storage or other infrastructural components. Usually the framework only allows for the development of stateless request-response driven applications, giving the cloud provider the possibility to simply spawn an instance of the application on any available machine as soon as a request comes in. This gives Platform as a Service applications a built-in scalability without the need for custom implementation of such features by the developer.

Compared to infrastructure as a service, the framework gives the developer less control over the system, while at the same time allowing for faster development of cloud-applications. Some responsibilities concerning security, scalability and availability are shifted from the developer towards the platform provider, making it easier to develop reliable services without much knowledge of the underlying infrastructure. Vendor lock-in is a major drawback of todays' PaaS offerings since no platform standard exists yet making it difficult to switch providers or spread an application across multiple clouds to maximize availability. The major PaaS providers today include *Google Apps*, *Microsoft Azure* and *force.com*.

Software as a Service (SaaS) delivers a ready-to-use software product to the customer which is accessed over the internet or the customer's local network. Processing and storage is handled by the cloud and the customer is oblivious to implementation or infrastructure details.

The offered products are rather off-the-shelf solutions than custom software. Some customization is usually possible especially in the enterprise market, but the intent of Software as a Service is to write a product once and sell it to many customers. This comes with a benefit especially for small customers who could not afford to have a complex system built for them and often don't employ an infrastructure that support such sophisticated security and backup solutions that a cloud provider can offer due to the economy of scale. They can take advantage of the high quality software that the provider needs to assure in order to attract large customers. The best known SaaS providers today are *Google* and *SalesForce*.

Besides these well-known service models described by the National Institute of Standards and Technology (NIST), a number of derived models exist to more precisely describe specific cloud offerings including *Data as a Service (DaaS)* and *Business Process as a Service (BPaaS)*.

Multi-tenancy

As stated before, cutting the service cost is a major goal of cloud computing. Cloud providers harness the economy of scale when serving a large amount of customers from a single infrastructure, and pass on some of their savings to the customer in order to create an incentive for moving into the cloud. A key technology for achieving lower operational costs is multi-tenancy. Without multi-tenancy, each customer has to be hosted on an individual machine making the cost of hosting a customer instance amount to the cost of operating a whole machine.

Scalability is not easily achieved in such an environment as well. So in order to prevent the application from becoming unresponsive due to overloading the machine, the hardware is dimensioned in such a way that it can meet the expected peak loads of the application. This results in the hardware being underutilized most of the time, bearing an average system load as low as 5 to 20 percent which leads to a very inefficient operation and high costs per request. [Fri11]

Multi-tenancy seeks to improve the hardware utilization and thus reducing the costs per request. This is achieved by running multiple application instances belonging to different customers on a shared physical machine. During idle times, a large number of tenants can share the same hardware so that their accumulative workload utilizes the machine in a more efficient manner. Since peak loads usually don't occur at the same time for every tenant, it is possible for a single machine to even sustain peak loads from a few tenants if dimensioned properly. Through means of virtualization, additional instances of client applications can be brought up or down on separate machines, depending on the current demand. This constitutes the elastic property of cloud computing, enabling it to provide the demanded performance just in time while still being able to efficiently utilize the available hardware resources.

Multi-tenancy can be used with various degrees of separation as illustrated in Figure 1.1.

A high degree of separation like setup *d)* provides better default security by using low-level isolation features of the Hypervisor, requiring an attacker to escape the Hypervisor in order to access data belonging to other tenants. On the downside, such deep separation introduces the overhead of a virtual machine, operating system and middleware stack to each tenant instance while a less separated approach like *a)* allows spreading this overhead over multiple tenants, resulting in lower per-tenant resource utilization.

Availability and response times can be more easily guaranteed in a less integrated setup since Hypervisors and operating systems provide mechanisms for assigning shared resources to individual instances. When using multi-tenancy on an application level, this sharing of

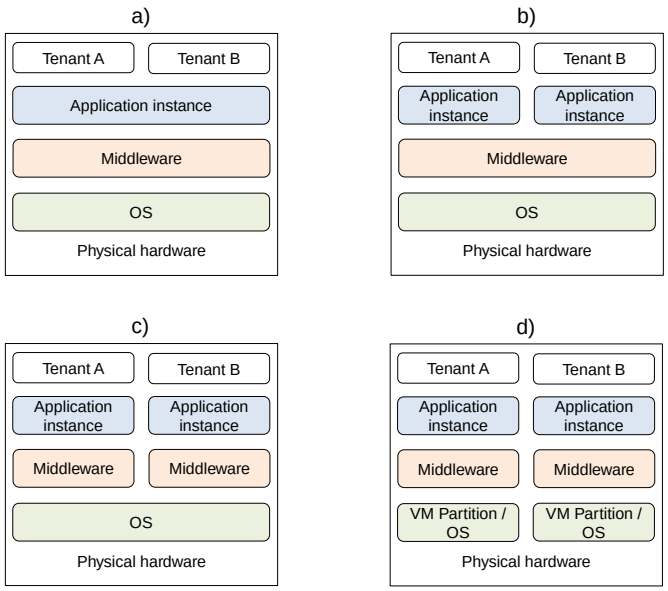

Figure 1.1: Degrees of Integration in Multi-Tenancy

resources has to be either done by the application or replaced by the automatic provisioning of additional instances.

The security implications and limitations of different isolation approaches will be more thoroughly discussed in Section 5.2. From an economic standpoint, lower degrees of separation are desirable in order to profit from the economy of scale introduced by the spreading of resource, maintenance and setup overhead.

1.2 Security Terminology

In order to have a common ground for discussion, the field of computer security and cryptography use a set of terms and phrases with often non-intuitive meaning. Some of the main recurring terms used in this thesis will be defined in the following.

Entity: In the process of authentication, an entity is the actual physical person requesting access.

Subject: Refers to the virtual representation of an entity in the context of authentication and authorization. A system is unaware of the actual entity but rather deals with a subject. Any entity can have multiple subjects, and a subject can be used by multiple entities.

Object: An object is a virtual or physical resource in the context of authorization. A subject may or may not perform certain operations on an Object.

Rule: A rule specifies which operations a subject may perform on an object.

Credentials: Also referred to as authentication data is a set of information used to identify and authenticate a subject. A user name and password or a certificate can be used as as credentials.

Key: Refers to a cryptographic key used in symmetric or asymmetric encryption.

Vulnerability: Any system may expose certain structural, operational, procedural or design specific weaknesses. These vulnerabilities are the points of entry for an attacker and may allow compromising the security of the system. [Amo94]

Attack: The exploitation of a vulnerability by an attacker is considered an attack on the system. The attack is always specific to a certain vulnerability and requires certain tools and access to the system or data to be successful.

Threat: Or attack vector is the combination of a vulnerability and a known or assumed attack. As long as there is no known attack that could exploit a vulnerability, arguably the threat remains low.

Risk: Refers to the likelihood of a vulnerability being discovered and an appropriate attack being feasible. The risk gives a unit of measurement about how dangerous a threat or vulnerability is and allows to prioritize security improvement efforts as well as assessing the security implications of design decisions and problematic requirements. As detailed in Section 1.2, using risk as sole measurement for security can lead to a seemingly secure system with hidden vulnerabilities waiting to be exploited.

Adversary and Attacker: The adversary is a hypothetical entity with a malicious intent to compromise the system. The term is often used in cryptography to objectively assess security by introducing an entity with idealized capabilities. Specifically the adversary is assumed to have physical access to the system and network and has any reasonable software or hardware tools available at his disposal. The adversary symbolizes the entirety of all possible attackers and allows discussing the security of a system on a design level without thinking of specific vulnerabilities or attack vectors.

An attacker is the actual entity perpetrating the attack by exploiting a specific vulnerability with a certain set of tools and capabilities. The relationship between these entities and concepts is described in Figure 1.2.

Security Principles

Based on different capabilities of an attacker there are different principles to securing a system that take into account possible attack vectors and provide security against a defined set of these capabilities.

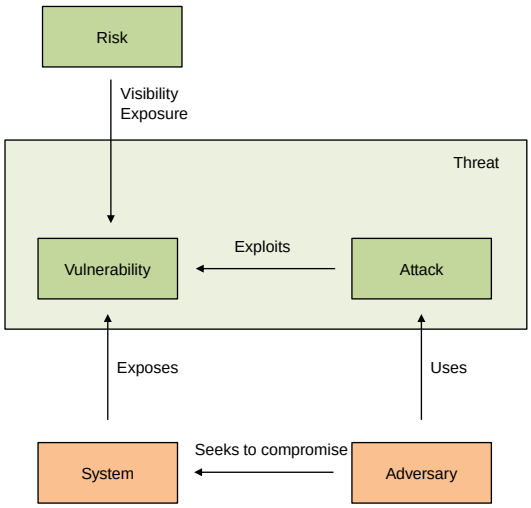

Figure 1.2: Security Terminology — Relationship

Shannon [Sha49] defined an attacker with unlimited computational capabilities and concluded that:

Definition 1. *If an information source conveys some information, then the attacker will surely extract that information.*

Requiring any theoretically unbreakable encryption to be as secure as the one-time pad (see Section 2.3). These requirements are arguably not practical for real world applications and don't reflect the reality of attackers capabilities. Diffie and Hellmann [DH76] revised these requirements in their 1979 paper on cryptography assuming an attacker has limited computational capabilities, laying the foundation for modern cryptography that relies on algorithmic complexity [Gol97]. This leads to the assumption made by today's cryptographers that:

Definition 2. *If an algorithm exists that would break the system, the attacker would surely find this algorithm.*

Resulting in a new model of an attacker in which he has limited computational capabilities but unlimited programming capabilities. This model appears to be confirmed by reality and the ongoing success attackers experience, and leads to the concept of open security.

Security by Design & Open Security

The rationale behind open security is that any algorithmic or design related weakness will eventually be discovered and exploited by an attacker, making the hiding of algorithmic and design aspects a futile attempt at securing the system.

The process of open security starts with designing a secure system while all the aspects are open to the public and the scientific community for peer review and evaluation. The goal is to have identified and eliminated all vulnerabilities and reached the consensus that the system is secure by design.

This approach is often used in the development of cryptographic algorithms like RSA which heavily rely on the open security aspect by publishing all design and algorithmic work for peer review [RSA83]. Security by design can be employed as well without the aspect of open security which is often preferred for commercial projects since the protection of intellectual property can only be guaranteed this way.

Security through Obscurity

In contrast to security by design, security through obscurity assumes the notion of Definition 2 to be false or at least exaggerated, allowing to secure a system by hiding known vulnerabilities and design related weaknesses [Pav11]. Not to be confused with security by design in closed systems, security through obscurity relies on certain aspects of the system being kept in secret since knowledge of these aspects would allow an attacker to break into the system.

This approach is especially popular with closed hardware systems since it is believed to be impractical and infeasible to reverse engineer the relevant aspects from a piece of hardware. This has been shown not to be the case with the 2008 crypto analysis of the *Mifare RFID Key Card* which managed to extract a shared secret key stored within the card by analyzing the physical structure of the microchip [KN08].

The still ongoing research of the *A5/1 Decryption Project* [1] managed to identify severe weaknesses in the proprietary A5/1 encryption algorithm used in GSM cellular phone networks by analyzing the protocol traffic and using common cryptographic tools. This attack relied heavily on the use of large amounts of computing power making it only recently possible for individuals or a dedicated group without large funding.

For the reasons stated above, security through obscurity cannot be considered a valid process for securing a system since its notion is an insecure system that tries to hide vulnerabilities from an attacker. It implies the possibility of surfacing these vulnerabilities with enough effort, leading to a calculated risk of the system being compromised. Therefore security through obscurity should not be applied as the only means of securing a system, especially in software projects with a broad user base that creates an incentive for attackers.

[1] A5/1 Decryption Project: https://reflextor.com/trac/a51

1.3 IBM SmartCloud Archive

To analyze security vulnerabilities and implement improvements, this thesis is based on IBM's enterprise cloud archive solution *SmartCloud Archive (SCA)*[2].

In order to offer the benefits of cloud computer to its enterprise customers, IBM launched its cloud computing strategy in 2007 by announcing software to manage public, hybrid and private clouds and beginning collaboration with customers and cloud providers like Google [Tomo7]. Beginning with the IaaS and PaaS solutions *SmartCloud Enterprise*, IBM launched the SaaS SmartCloud product line in 2011. As one of the first products to be launched in this context, SmartCloud Archive offers enterprise content management in the cloud.

In order to comply with government regulations, enterprises have to keep archived copies of documents and data for certain periods of time. Enterprise content archiving has to enforce retention policies to prevent the deletion of objects and must allow e-discovery to locate relevant documents for legal processes. Content classification and attribute management play a key role in achieving these goals so SmartCloud Archive choose to use an enterprise content management system as a core component to store documents. An exhaustive insight into enterprise content management systems and design aspect of SmartCloud Archive is given by [Bö11] and [Fri11].

Since SmartCloud Archive (SCA) stores sensitive customer data in the cloud, data security is an important selling point and basic requirement for any enterprise cloud solution.

The lifecycle of customer data in SCA begins with data being uploaded into the system and ingested by either the *SCA Batch loader*, *IBM WorkplaceXT* or the *IBM Content Collector ICC for Email*. The data then resides within the *IBM Filenet P8 Content Engine*, waiting to be either exported for a legal case through *eDiscovery Manager* or disposed of after its retention time by a disposition sweep.

The architecture of IBM SmartCloud Archive is comprised of a set of IBM enterprise products and custom components combining these into an integrated cloud solution. The storage layer is provided by the *IBM General Parallel File System™(GPFS)* on top of which the *IBM FileNet P8* enterprise content engine manages the customers data. An *IBM DB2®* Database instance stores repository meta-data and most of the IBM SmartCloud Archive configuration. The *IBM Tivoli® Directory Server (TDS)* stores user accounts for the integrated system users and serves as an authentication provider. Synchronization with a customer directory server can be used to support users already registered within the customer's directory.

The *IBM Websphere®Application Server* serves as a platform for the custom components *Batch Loader* and *IBM SmartCloud Archive Portal* while *IBM eDiscovery Manager* and *Workplace XT* provide access to the repository content. Archive statistics and usage reports are provided by the enterprise analytics tool *IBM Cognos*. Figure 1.3 shows an overview of the architecture components and their relationship.

[2] https://www-935.ibm.com/services/us/en/it-services/smartcloud-archive.html

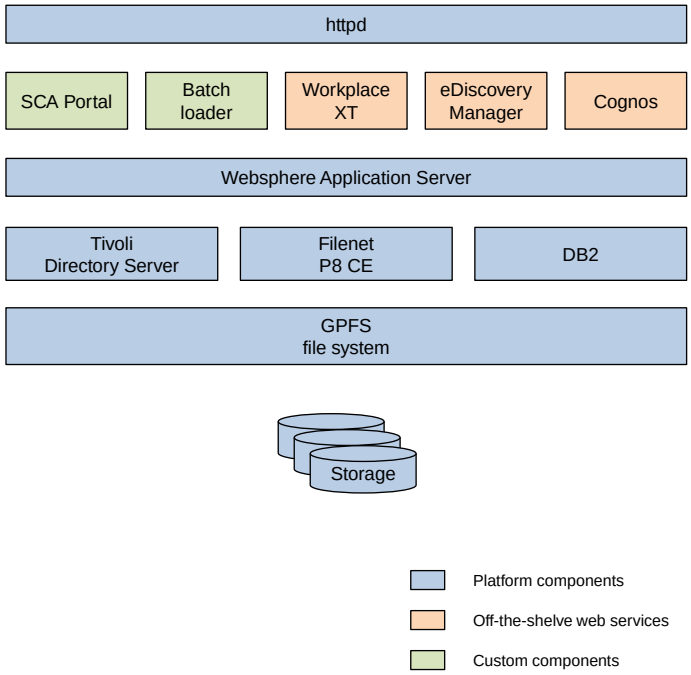

Figure 1.3: IBM SmartCloud Archive Components

2 Cryptography

2.1 Introduction

The intent of encryption is to reduce the amount of data that has to be hidden from an attacker in order to prevent disclosure. A large amount of encrypted data can be stored or transmitted in an insecure environment while only a small encryption key needs to be kept secure, lowering the required security standard for large portions of the involved infrastructure. This shift of sensitivity from data to encryption keys introduces the field of key management discussed in chapter 4.

Data in information systems exists in two states, data at rest and data in motion. Security concerns in both states can be met by encrypting the data at rest and transferring it only in its encrypted state or by utilizing an additional encryption layer for wrapping the data when it is being transferred. This yields two mechanisms for dealing with encryption on a holistic level.

Encryption oblivious cloud describes the scheme of encryption the data only once at the source. This encrypted data can be handled by any type of trusted or untrusted information system and is only decrypted by the owner once he retrieves it. This mechanism has one major drawback in the fact that it is no longer possible for the information system to do any type of operation on the data except storing and replicating it.

Encryption aware cloud can be used when dealing with a trustworthy information system which should be able to operate on the data. Encryption is employed for data at rest while the encryption key is known to the information system. The system can decrypt the data in order to perform operations such as indexing or database operations and encrypt the data again before it is committed to storage. When transferring the data, a new encryption layer is introduced for securing the connection.

Depending on the type of service that is used, an *Encryption oblivious cloud* may not be a feasible option and an *Encryption aware cloud* is required. A certain degree of trust must be put in the service provider when using this mechanism, while the encryption schemes discussed below combined with a secure key management concept discussed in chapter 4, give service providers the ability to prove the trustworthiness of their solution.

2.2 Encrypting Data

Data at Rest

The goal of encrypting data at rest is to prevent an attacker from gaining access to plain text data under certain conditions, and prevent or detect manipulation of data to guarantee data integrity.

Preventing and Detecting Data Manipulation

Preventing and detecting data manipulation seeks to counteract the integrity threat discussed in chapter 1. Detecting data manipulation is a weaker requirement than preventing it, since prevention is only possible if any malicious access can be ruled out.

Manipulation detection through digital signing uses the signature approach discussed under *asymmetric encryption* below to identify the specific content of a document or file by calculating its hash value and encrypting this value with the private key of a digital certificate. This digital signature is then stored together with the data to allow verifying the content by decrypting the hash value with the public key and comparing it to a new calculated value. This approach is well suited for static content and archiving or legal purposes.

Manipulation detection through hash chains is a novel approach to verify the contents of log files and journals. Its most prominent use can be found in the distributed version control system *git*[1] where it prevents the injection of malicious entries. All these applications have in common that they only append entries to a file or repository but never change old entries. Hash chaining starts with a root entry of which a hash value is calculated and appended to the second entry. The end of this chain is always a hash value of the row (entry n, hash (row n-1)) as shown in Figure 2.1 [BS99].

This approach follows the rationale of encryption in reducing the amount of sensitive data. Hash chaining reduces the amount of data that needs to be stored, and calculated in order to detect manipulation. It is tailored to storage systems that only allow appending data and therefore makes use of the fact that old entries never change. This makes it possible to only compute hash values for new data and include the existing hash value of the old data in that calculation.

The hash chain starts empty, once an entry is inserted the corresponding hash value of that entry is calculated and written to the next row. This last row of the chain only contains a hash value of the previous row. When the next entry is inserted, it is appended to the hash value of the last row and a new hash value for that row is calculated, comprising the new last row. This way, the last row of the hash chain always contains a hash value capable of recursively verifying the whole chain.

[1] http://git-scm.com

In order to detect manipulation, a copy of that value must be stored in a secure location after each append operation.

Instead of securely storing the last value, manipulation detection is also possible if instead of a hash function, a *Message Authentication Code (MAC)* is used. The MAC is a hash function that accepts a secret key as a second input parameter. It calculates a hash value based on the input, and the secret key. An attacker who tries to manipulate the hash chain is not able to compute legitimate hash values without the necessary secret key.

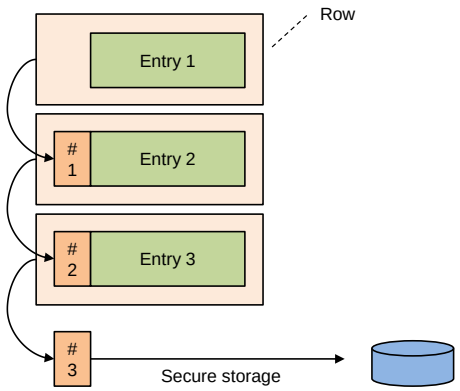

Figure 2.1: Hash Chaining Log Files to Detect Manipulation

A common standard for encrypting data at rest is yet to be established. All proposed mechanisms are either proprietary in nature or part of ongoing research. The IEEE[2] launched the standardization project *P1619* towards that goal and commissioned the *IEEE P1619 Security in Storage working group*[3]. Their approach is to find a holistic solution for storage encryption that includes encryption algorithms and methods as well as key management. The findings of this working group will be discussed at the appropriate sections in this thesis along with other solutions.

The encryption of data at rest can be implemented at any level from the file itself down to the whole storage medium. Figure 2.2 gives an overview of the storage system and shows at which point encryption can be employed. While solutions *a* and *b* rely on hardware assisted encryption, *c* through *e* only require encryption software and can be used with any hardware storage system. Solutions *c* and *d* utilize operating system or file system components to offer transparent encryption while solution *d* refers to internal application level encryption. Different hardware and software based products will be discussed in the following.

[2] http://ieee.org
[3] http://siswg.net

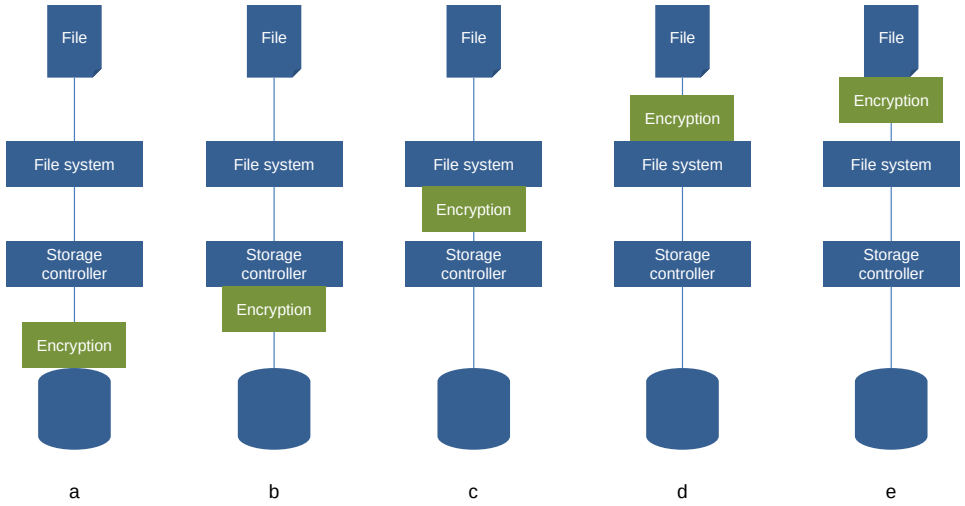

Figure 2.2: Data Encryption Points Between Hardware and Software

Encrypting Drives and Partitions

Hardware based Data encryption at the lowest level, the storage device, can be separated further into two different approaches. *External encryption* as displayed in Figure 2.2.a utilizes a hardware encryption module that sits between a regular hard drive and a storage controller while *internal encryption* (called Discryption, Figure 2.2.b) integrates the encryption module directly into the hard drive [Haro7].

External encryption has an inherent drawback; it allows an attacker who gains physical access to the drive full access to the cipher text stored on the device. By comparing multiple images of the cipher text an attacker can derive information about access pattern and recently changed files in order to manipulate data or the behavior of the system in some way. An attacker might be able to undo certain operations by restoring data from on older version of the cipher text. Knowledge about the position of system files that can be gained by e.g. statistically analyzing the changes to the cipher text after a system update was performed, can be harnessed to change the behavior of the system. The content and layout of common system files can be easily obtained and allow an attacker to randomize a jump address within

that file by simply changing the corresponding part of the cipher text and thereby rendering certain security or other mechanisms of the systems ineffective. [DMLW08]

Internal encryption combines encryption with authentication and allows only a valid user in possession of the key access to the data. Without the key, not even access to the cipher text is possible, rendering the aforementioned attacks futile. It is however possible for an attacker in physical possession of the drive to bypass the integrated controller and gain direct access to the platters by employing data recovery tools and equipment. Such a procedure would require the device to be brought to a special facility for a prolonged period of time and would leave visible evidence of the process on the device. This makes manipulating the system and data virtually impossible since these attacks rely on the process going unnoticed. Attempting to decrypt the data by brute force or cryptographic analysis would be the only viable option for an attacker in this case. Internal encryption follows a *Full Disk Encryption (FDE)* approach which puts the contents of the whole disk into one security context by encrypting the whole drive with one encryption key. While this suits the use cases for personal computers and on-premise servers it does not provide any separation in multi-tenant cloud computing scenarios. Some external hardware based encryption systems mentioned below are capable of creating multiple security contexts within the same disk or array by encryption on the partition level. This can be a sufficient level of separation in some multi-tenant scenarios but the low-level process of creating partitions and the limitation in the amount of possible partitions makes this approach unsuitable for massive multi-tenant systems.

Software based Software based approaches to drive or partition encryption are more widespread than hardware encryption and since no additional hardware is required, cheaper solutions are possible. Concerning data security, they suffer the same weaknesses as external hardware based encryption. Additionally they are open to a set of new risks since the encryption software and the memory containing the key are not located in a secured piece of hardware but in the more easily compromised environment of the operating system, making it susceptible to the same kind of attacks used on other software. Encryption keys stored in memory can be read by other privileged processes and the software itself can become a target to manipulation [ASR09].

In environments that use virtualization, software based disk encryption can be applied at, or below the hypervisor in order to hide the encryption layer from the virtualized operating system and software stack. [LC10] has demonstrated an approach to hypervisor assisted full disk encryption that has shown additional protection against attacks originating fron within the virtualized environment. Compared to hardware assisted encryption, this approach is only as secure as the separation provided by the hypervisor.

Data in Motion

Encrypting data in motion is a widely used mechanism today and thus fairly easy to implement. It can be employed on different layers of the OSI protocol stack[4] in order to achieve different goals. The most straightforward way of encrypting a connection is to tunnel all traffic through a virtual priate network (VPN). A VPN has two endpoints between which an encrypted communication channel is established over the Internet or any untrusted network. The endpoints can be in form of a hardware appliance or a piece of software. Most communication protocols can be tunneled through the VPN without additional considerations making this approach a good choice for retrofitting security onto a system.

Deeper integration of encryption into the communication protocol allows for easier service deployment since no additional VPN infrastructure needs to be implemented on client side. Another way of securing a non secure communication protocol is offered by the SSL/TLS protocol family. These protocols encapsulate the entire data stream of a layer 5 protocol between the client and server. While a VPN connection usually does not only provide encryption of the data stream but also authentication of the user, SSL/TLS traditionally only provides data encryption and leaves authentication and authorization to the application developer although such functionaity is provided by these protocols.

2.3 Encryption Algorithms

Cryptographic Principles

Cryptography always involves two stages, encryption and decryption. Plain text is encrypted into cypher text and vice versa. We talk about **symmetric encryption** when both de- and encryption key are identical or trivially related. In **asymmetric encryption**, those two keys are not identical, and non-trivially related which makes decryption impossible with only the knowledge of the encryption key. Symmetric encryption is much more efficient than asymmetric encryption which makes it the preferred method for encrypting bulk data. Applications for asymmetric encryption are mostly found in key exchange or identity verification scenarios and will be discussed in detail after symmetric encryption.

The One-Time Pad

The one-time pad is an idealized cryptographic key that is used as a matter of discussion and comparison with other encryption keys and their security under certain conditions. The one-time pad is as long as the message it encrypts, and must be non-repetitive and truly random. This results in a cipher text that is truly random as well and allows an attacker no

[4] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.9497&rep=rep1&type=pdf

insight into the message content without knowledge of the used one-time pad. Therefor the one-time pad provides perfect cryptographic security under any condition which makes it an ideal model for describing the security of other encryption keys.

Modes of Operation

In data encryption, two *modes of operation* can be identified. **Block ciphers** are limited to encrypting a single fixed-length block of data at once [Ando1]. They derive a secret key from the provided encryption key and perform a transformation using the plain text and secret key as input. The output of this transformation comprises the cipher text. In order to encrypt larger amounts of data this transformation has to be sequentially applied to each consecutive block of plain text. This can be done in different ways; one approach is called an electronic codebook (**ECB**). This straight forward encryption scheme uses the same secret key for each block of data and thus yields the same cipher text for plain text blocks containing identical information, making it vulnerable to analysis since it reveals data patterns. A more sophisticated approach can be found in the class of cipher-block chaining (**CBC**) algorithms. These use the cipher text output of the previous block as a modifier for the secret key and use this new key to encrypt the next block of data, resulting in pseudo-random cipher text that hides data patterns as demonstrated in Figure 2.3. Image *a* shows the plain text input data while *b* is an image of the encrypted cipher text using an electronic code book algorithm. Image *c* applies the same secret key but uses a cipher-block chaining approach to encrypt the data. Both examples use the same 256 bit AES encryption algorithm to generate the secret key from the pass phrase *"123456"*. Note that the pass phrase doesn't influence the pattern preserving nature of ECB encryption. A different pass phrase would only result in a different coloring in this visual example.

In order for the chain of encrypted blocks in CBC to start, an initialization vector (IV) is used in combination with the secret key as input for the first block. Repeated use of the same IV under the same key leads to the CBC scheme behaving like ECB for the first block of cipher text, allowing an attacker to observe similarities in the transmitted message. The IV is for that reason always required to be pseudo-random in order for it not to be reused, or allow conclusions about the message. Depending on the used encryption algorithm, the IV may also be required to be unpredictable to an adversary. The AES algorithm used above only requires a pseudo-random IV which allows it to be a nonce (number used once) which is generated and stored or transmitted with the cipher text.

Stream ciphers provide a method for encrypting a continuous stream of plain text data without separating it into blocks. Anything from a single bit to an arbitrarily long stream of data can be encrypted on the fly. Block cipher algorithms can be used in a similar manner by using the modes of operation described above which implies some drawbacks. Since these algorithms have a fixed block length that must be encrypted at once, they impose a certain delay on streaming applications while they collect data to fill a block. Small bursts of data that don't fill up a single block still result in a cipher text of full block length imposing an overhead on the amount of produced cipher text.

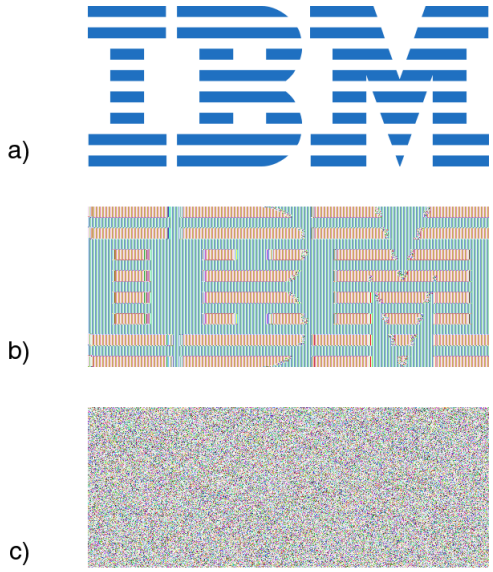

Figure 2.3: ECB and CBC Encryption

For these reasons the class of stream cipher algorithms is preferred in applications that require true streaming data encryption like communication. Dedicated stream cipher algorithms use the secret key as a seed for a pseudo random number generator and combine this stream by means of a logic function with the plain text stream, resulting in a stream of cipher text. By inverting the logic function the same mechanism can be used to decrypt the stream which reduces the complexity of hardware or software implementation of stream ciphers. This simple mode of operation makes stream ciphers very fast in hardware and software implementations but makes them more vulnerable to attacks as well [And01].

For reasons discussed in Section 1.2, trust in the security of encryption algorithms is strengthened by evaluating them through the scientific peer review process. This is why most established and widely used encryption algorithms are freely available to the public. Table 2.1 shows the most common encryption algorithms today, their operational parameters and the result of extensive cryptographic analysis conducted since their publication. The block cipher algorithms considered most secure by today's standards are AES and triple DES. The most successful crypto analysis of AES available today allows a four times faster decryption than using a brute force attack, reducing the number of required operations to 2^{190} for 256 bit AES [AB11]. The triple DES standard, a successor to DES originally developed by IBM in 1977, has been shown to be susceptible to known-plaintext attacks and can be broken with 2^{88} operations. A performance comparison of triple DES and AES shows that AES outperforms triple DES in most applications, making it the encryption algorithm of choice for general purpose encryption [NJ05].

Encryption algorithm	Type	Key size	State/Block size	Speed	Security
AES	block	128, 192 or 256 bits	128 bits	low	very high
DES	block	56 bits	64 bits	high	very low
Triple DES	block	56, 112 or 168 bits	64 bits	low	high
Blowfish	block	1 - 448 bits	64 bits	high	high
RC4	stream	40 - 2048 bits	2064 bits	very high	low

Table 2.1: Comparison of Symmetric Encryption Algorithms

Hash Functions

The third pillar of cryptography besides asymmetric and symmetric encryption is comprised of **hash functions**. They provide a way to detect data manipulation and verify integrity by calculating a hash value over a given data input (message). The two main properties of cryptographic hash functions are collision resistance and inversion resistance. Collisions occur when different messages produce the same hash value. This characteristic can't be avoided since hash functions accept arbitrarily large messages but always produce a hash value of fixed size. Collision resistance should provide an even distribution of hash values for each domain of similar sized messages and make the intentional computation of a colliding message for a given hash value infeasible. Inversion resistance is similar to collision resistance in that it guarantees that the computation of an inverse function to a given hash algorithm is infeasible. Computation of such a function would allow to easily find a message for any given hash value.

With these properties hash functions are used to verify the integrity of a message by storing its hash value in a safe location or encrypting it and later calculating and comparing the hash value again.

They are widely used for authentication purposes by storing hash values of subjects credentials in the identity service and comparing it. An attacker gaining access to the content of the identity service can't reproduce the credentials from the stored hash values and is unable to use this data for identity theft purposes.

Asymmetric Encryption

Contrary to symmetric encryption, asymmetric encryption uses different keys for de- and encryption. Asymmetric encryption therefor requires a *key pair* rather than a single key. The key pair consists of a public, and a private key. Both can be used to encrypt data which is then decrypted by the other key. This allows the possibility of establishing a secure communication channel over an insecure transmission channel by both parties exchanging their respective public which is used to encrypt the data being transmitted. An eavesdropper can only gain possession of the public which doesn't allow him to decrypt any of the data being transmitted. Due to the inefficiency of asymmetric encryption algorithms, this scheme

is often used to mutually generate and exchange a symmetric encryption key which is used for further communication.

Besides key exchange, asymmetric encryption is applied in the digital certification process for identity verification and integrity assurance as described in section 3.1.

3 Authentication and Authorization

Authentication is the first line of defense in securing systems against unauthorized access. When accessing a service, a user or process is challenged for credentials which allow the system to decide whether the request should be granted or not. The security of authentication mechanisms therefor relies on the credentials being secret.

3.1 Authentication

Authenticating Users

The process of authentication allows a server to verify the claimed identity of a subject in order to determine access rights to the system. The rationale behind authentication is that the subject is in possession of a *private secret* that no one else knows. The subject proves his identity towards the server by proving he is in possession of the private secret. This requires the server being able to *verify the secret* provided by the subject. Since the private secret is no physical object but plain information, the term *in possession* is ambiguous in that indistinguishable copies can be made and used by an attacker for authentication. Mitigating the threat of copies is achieved by designing the authentication method in such way that it *doesn't include creating any copies* of the secret for either the server, or during the authentication process.

The three requirements for authentication methods are therefor:

- The subject has a private secret
- The server can verify the secret
- No copies of the secret are created

While the first two are basic structural elements required for the function of the authentication method, the third requirement improves the security of the method by preventing abuse.

The following section will discuss different approaches to authentication and present commonly used methods that implement the above three requirements in different ways, leading to security and system complexity implications.

Certificate based Authentication

Public key certificates are a mechanism for subjects to prove their identity to a service by stating that identity within a certificate which is signed by a third party. The service trusts this third party to have verified the identity of the subject and can then verify the certificate for integrity and challenge the subject to prove he is the owner of the certificate. Through that process it is possible to authenticate a subject without the service having any prior information about it other than trusting the third party who signed the certificate.

Certification Process

The certification process is separated into three steps; certificate creation, signing and distribution.

First, certificates are created by the subject and the trusted third party, called the *certificate authority (CA)*. Creating a certificate begins with generating a public/private key pair for asymmetric encryption. Then a certificate is created which can be considered a plain-text file containing information about the identity of the subject, and its public key.

The signing process starts by delivering the certificate to the CA who then verifies the identity claimed in the certificate. The CA inserts its own identity information into the certificate and subsequently calculates a hash value of the entire certificate (identity information and public key of the subject, identity information of the CA). The certificate is signed by encrypting this hash value with the CAs private key, and appending this signature (encrypted hash value) to the certificate.

It now contains:

- Identity information about the CA
- iIdentity information about the subject
- The subject's public key
- A hash value of the above information, encrypted with the CAs private key.

The certificates are now being distributed among service providers and subjects. The subject receives his own certificate and service providers receive the CAs certificate. the subject can now authenticate against service providers by sending his certificate, which is validated by the service provider through calculating the same hash value the CA did, and comparing it by decrypting the hash value within the certificate with the CAs public key.

To verify that the subject claiming to own the certificate is in fact the owner, the service provider initiates a challenge-response mechanism in which he calculates a secret random number and encrypts it with the subject's public key found in the certificate. This challenge is then sent to the subject who decrypts the number using his private key and responds to the challenge with the random number in plain text. The service provider compares the

subject's response with his generated random number and now has proof of the subject's identity.

This scheme assumes that only a legitimate subject is in possession of the private key belonging to a certificate, while the certificate itself may be public.

Public Key Infrastructure In certificate based authentication, a public key infrastructure harnesses the transitive nature of signed certificates to create a tree-structure with chains of trust extending from a root certificate authority to any signed certificate below.

A PKI involves the root certificate authority (CA) which is considered a trusted third party by both the server, and client. The CA assures the validity of any certificate it issues by verifying the identity of the subject requesting certification. The public key certificate of the root CA is distributed among all participants and allows them to verify the certificate of any other party.

This scheme is widely used in online server verification through the SSL protocol. Client web-browsers contain a copy of the root CAs certificate which allows them to verify the identity of a web server before establishing encrypted communication.

Due to the transitive nature of signed certificates, any subject holding a certificate can itself issue and sign new certificates. Clients can recursively verify the integrity of these certificates up to the root CA and decide whether to trust the certificate based on the signing parties involved in the chain of trust.

Authenticating Systems and Processes

System and process authentication is required when services in a distributed environment interact in order to assure that only legitimate requests are executed. Similar to a user request, a request originating from a service must be assigned to a subject, and that subject must be authenticated since an attacker can forge system requests which would allow unauthorized data access.

Contrary to users, system can be configured and programmed to only issue legitimate requests, providing a basic level of security given that the systems are not compromised. Section 5.3 will demonstrate that many attack vectors against cloud services rely on unauthorized system-to-system request

Deep Authentication

With deep authentication the security issues with high-level multi-tenancy discussed in section 5.2 can be mitigated.

Tenant separation on a high level is desirable for performance and maintenance reasons but it raises the tenant isolation layer higher in the application stack, bringing it closer within the reach of an attacker.

Low-level tenant isolation is favorable from a security standpoint since the tenant isolation is hidden under application stack layers that an attacker would have to penetrate.

As section 5.2 has shown, each application stack layer provides methods for tenant isolation that can improve the security of a system. An attacker has to either penetrate a layer with tenant isolation until he reaches a non-isolated layer, or find software vulnerabilities that allow him to circumvent the isolation. Deep authentication allows to make use of isolation mechanisms in each layer of the application stack by using tenant-specific authentication in each layer.

3.2 Authorization

The process of authorization takes place after a subject is authenticated and determines which operations on the system shall be granted or denied.

Authorization mechanisms are found in operating systems to control access to objects like files and resources, and are used in any kind of multi-user distributed system. [SS94]

A basic building block of authorization mechanisms are **Access Control Lists (ACL)**. An ACL is attached to an object or a group of objects, and contains entries for users or groups of users stating the permissions to operate on that objects. An ACL for a file object would contain the name of a user and state if that user has permissions for reading or writing to that file.

The disadvantage of plain ACLs is their verbosity and the complexity of determining a user's effective permissions over a large number of files which is only possible through examining the ACL for each file.

To mitigate the verbosity of ACLs UNIX systems have introduced an authorization mechanism based on ACLs which is known as **Discretionary Access Control (DAC)**.

In DAC, each object has an owner who is the creator of the object, or another subject appointed by the previous owner. The ACL only contains three entries. Permissions regarding the owner, a group and any subject not belonging to the first two categories. This authorization mechanism is discretionary in that it is at the discretion of the object creator or owner to define the access permissions.

With DAC the disadvantage of determining the effective permissions of a subject is retained from ACLs while the verbosity is eliminated by limiting the ACL to the aforementioned three entries at the cost of losing flexibility.

A different approach is taken by **Mandatory Access Control (MAC)** which makes use of a large central access control list. The access permissions are mandated by a central authority

that is the sole entity with the right to modify access permissions. Subjects may have to right to create files but they don't assume ownership over them and can't decide over their access policy.

The advantages of MAC are that subjects should be given access to objects without having the right to pass that permission on to other subjects. MAC also eliminates the disadvantage of ACL and DAC in determining the effective permissions of a subject by having only a single, central access policy that can more easily be analyzed and managed.

A concept found in distributed multi-user systems that deal with operations and access to sub-systems rather than objects and files is **Role Based Access Control (RBAC)**. In RBAC a set of roles is defined where each role specifies a list of access permissions to the certain system operations. These roles are static in nature compared to the other mechanisms up the point where only a predefined set of roles is available. Users or groups of users are now assigned to roles giving them a set of permissions that can easily be deduced later by the user's role assignment. [BEE+10]

Certificated based Authorization

Certificate based authorization is a method of utilizing certificates not only for the purpose of authenticating an entity, but also for authorizing or denying requests. Contrary to authorization methods based on access permissions residing on the server authorizing a client's request, the concept of certificate based authorization allows each client to provide his own access permissions which are then evaluated by the server.

Through the technology of signing certificates by a trusted authority, it is possible for the server not only to verify the identity of the subject, but also verify the integrity and legitimacy of additional information contained within the certificate. In the process of issuing a signed certificate, the certificate authority verifies the identity information provided by the subject and attests the validity of that information with its signature.

For certificate based authorization, the CA also attests the access permissions belonging to the subject by including them with the certificate before signing. To verify the permissions, the CA cooperates with the party who is affected by them by either having this party present during the certification process, or by issuing the certificate to that party on behalf of the subject.

Managing the access permissions also involves revoking access rights from clients. In certificate based authorization this is done by invalidating the entire client certificate through Certificate Revocation Lists (CRL) which are a well-supported feature in public key infrastructures. Since each certificate usually has an expiration date after which is becomes invalid, entries in the CRL must only be kept until their natural expiration date, keeping the CRL from getting bloated over time.

The advantages of such an authorization mechanism over server-side access control lists lie in the separation of duties, and the conformity with large-scale distributed systems.

It allows the separation of duties between the server or infrastructure provider, and the service provider or data owner. Without direct interaction with the server, an administrator can issue certificates granting certain access permissions to a client. In a distributed system these can be evaluated by any server without the need to synchronize access permissions across a large number of systems. In use cases with a very large number of clients and complex access patterns, this concept reduces the amount of information that needs to be stored and managed by the server as well as the number of entities who need administrative access to the server in order to manage the access control lists.

4 Key Management

4.1 Introduction

Key management systems play an integral part in computer security concepts especially in a multi-tenant cloud environment where resources are shared between customers.

In order to utilize the security mechanisms like authentication and encryption that were discussed in this thesis, three supporting technologies were identified that constitute the backbone of any holistic security architecture. These are **Identity Management**, **System Credential Management** and **Encryption Key Management**. Not all of these are responsibilities of a key management system in the conventional understanding. The topic of identity management is covered by directory servers like the IBM Tivoli Directory Server used in IBM SmartCloud Archive while system credential management and encryption key management are similar tasks which lack widespread product support.

A 2011 survey by the *European Network and Information Security Agency (ENISA)* among enterprises and government bodies that employ encryption has shown that the topic key management is not well addresses and often neglected. It was found very common that sensitive encryption keys or certificate keys are found within the operating system and included in unprotected backups.

As described in [Amo94] the security of a system relies on securing the three operational aspects; data at rest, system operation and communication. The following will describe the tasks of the three aforementioned systems and analyze their security requirements considering these aspects.

The use cases for all three tasks can be satisfied by a system following the client-server architecture, while both of these parties have different security requirements.

Identity management

Consists of tasks related to user authentication, authorization and permission or role management. These services are often provided by a directory service which is well suited for large organizational structures involving large user bases and sophisticated synchronization and replication mechanisms.

Security of data at rest:

Identity management systems need to be able to identify a certain subject and authenticate its user credentials, no sensitive data needs to be stored in plain text. By using hash values to verify passwords an attacker gains little information from unauthorized data access to the stored information of an identity management system. Data manipulation on the other hand can compromise the security of the system more drastically since an attacker can modify the stored password hashes to a value known to him and adapt the authorization rules to gain unlimited access to any system relying on the identity management

Security of operations:

Since a client application can't verify the information it receives from the identity management system, the correctness of internal operations must be guaranteed. Restrictions and policies are only enforced programmatically and not cryptographically. The logic of the identity management system must adhere to its rules and not expose any vulnerability.

Security of communication:

If an identity management system is configured to enforce restrictions on password complexity it must receive a plain text password in modify and create operations. Authentication and authorization operations do not require any plain text credentials to be submitted and therefor don't expose valuable information to eavesdropping attacks. Man-in-the-middle attacks must be avoided in all communication cases since they allow an attacker to forge any response of the system and grant any authentication or authorization request.

Encryption key management

Is the primary domain of key management solutions and it's responsible for securely storing cryptographic keys used in various encryption applications. Content stored in a key management system are cryptographic objects as discussed in chapter 2 like symmetric encryption keys, asymmetric key pairs and public key certificates. The task of a key management system is to store and manage these sensitive objects and provide a secure way of accessing them.

Different approaches are possible to fulfill these requirements. Since the sensitive data does not require external verification and is only used locally, the involvement of a server is not necessary from a pure functional point of view. The cryptographic keys can be stored and managed locally, and be protected from unauthorized access through operating system and software measures or a hardware based approach.

From a security standpoint the local storage of encryption keys has some drawbacks. An attacker with physical access to the machine could circumvent software or hardware protection measures and access both the encrypted data, and the respective encryption keys. Such an attack could be detected, but not remotely prevented once the attacker is in possession of the physical hardware.

A centralized client-server architecture for a key management system provides not only physical separation of the encrypted data and encryption keys, but adds the possibility of centrally managing access permissions and revoke the rights of compromised clients.

Due to these security improvements provided by a client-server approach, such a system is preferred and will be the basis of further analysis in this thesis.

Security of data at rest:

Due to the design of a key management system, all sensitive data is encrypted and does not reveal any valuable information to an attacker. Public keys and certificates are often stored in plain text depending on the use case text since they are considered non-critical information.

Access to the content of the key management system is therefore not critical while the necryption key used to secure the content is considered highly sensitive information. This *master key* needs to be well protected since the security of the whole system relies on it.

Security of operations:

An idealized key management system would secure operations on the server cryptographically and would not have to rely on programmatically enforcing policies and rules in order to be secure. Such a system would have to utilize the client provided credentials as encryption keys for the content in order to render all information stored on the server non-sensitive on itself. If multiple clients are associated with the same stored objects, multiple copis would have to be created and each encrypted with the key belonging to a client. More sophisticated encryption schemes could be applied in order to reduce the amount of redundancy and management overhead.

Operational requirements can make this approach infeasible and lead to a key management system that has to rely on the correctness of the program.

Security of communication:

The communication with a key management system needs to be secured since sensitive cryptographic material is transferred from and to the system in most of its operations.

System credential management

Similar to encryption keys, system credentials are used by an application to acquire data or services from another system which is usually a middleware component like a database. This information is often stored in configuration files since it is required for basic application functionality. Some key management solutions like *IBMs Tivoli Key Lifecycle Manager (TKLM)* support the storing of arbitrary *secure data* or credentials in special and may be used as a credential management system as well.

Security of data at rest:

Depending on the credential management system in use, credentials may be stored in plain text or encrypted form. Appropriate security measures discussed in section 4.3.

Security of operations:

Depending on the use of plain text configuration files for storing credentials or the use of encrypted storage, different requirements occur. In the plain text scenario, credential management poses the same security requirements as an identity management system. Credential management systems with encrypted storage expose the same requirements for operational security as a key management system.

Security of communication:

The same requirements as in a key management system apply since all information transmitted to and from the credential management system is considered critical.

Conclusion

The role of identity management creates a different set of requirements for these systems than in key or system credential management. Identity management and verification per definition involves a third party attesting the identity of one subject to another, necessitating the use of a client-server approach or public key infrastructure as described in Section 3.1.

System credential and encryption key management systems share the same characteristics in that they store and manage sensitive information that must be made available to client systems but not attackers, leading to the common requirement of authentication against such a system and an authorization mechanism for granting or denying requests.

Assuming a client-server based architecture for all three types of services, and server side encryption of the sensitive information, table 4.1 states the sensitivity of data at rest and in operation on the server as well as during communication with the client.

Description of terms used in table 4.1

○ The information may be read and altered by an attacker without compromising the system.

◐ The information may be altered by an attacker, but not read.

◑ The information may be read by an attacker, but not altered.

● Any kind of access to the information may compromise the system.

Key management task	Data at rest	System operation	Communication	
			To Server	To Client
Identity	◑	●	●	◑
Encryption key	○	●	●	●
System credential	○	●	●	●

Table 4.1: Key Management Tasks — Information Sensitivity

Concluding that both a key management and a system credential management system share the same security requirements and basic functionality, a system for both roles can be designed. The security requirements allow the stored data to be considered non-critical which allows makes the handling of backups and storage more efficient since no security protocols have to be followed. As stated above, the only sensitive information that needs secure storage is the master key used to encrypt the stored keys or credentials.

The operations of such a system need to be programmatically safe since has been shown that cryptographic safety is not feasible. Secure communication is necessary due the sensitive nature of the data being transmitted. Conventional methods for securing communications described in Section 2.2 are sufficient.

4.2 Key Management Concepts

Since the task of managing keys and credentials can be handled in different ways, the following will analyze their security implications and point our recommendations for a secure system design.

In the system architecture, a physical and a logical layout can be distinguished. While the physical layout addresses the distribution of client and server roles and their physical security requirements, the logical layout defines the separation of duties between the components.

The goal of choosing the physical layout beyond that of a single machine is to prevent an attacker from gaining access to both the encrypted data, and the encryption key.

Different architectures comprised of a physical and logical layout can be grouped by a few characteristics listed in Table 4.2. The encryption keys can be stored either locally on the

hardware holding the encrypted data, or remotely on a centralized key server. Such a key server requires some kind of authentication to verify key requests, creating the requirement of storing authentication credentials on the machine operating on the encrypted data. This approach follows the rationale of encryption by further reducing the amount of sensitive information from the encryption keys in the case of local storage, to authentication credentials in the case of a remote key server.

Both approaches have in common that the client machine (operating on the encrypted data) needs to store some kind of sensitive information, be it encryption keys or authentication credentials. This further groups key management system by the type of protection in place to secure this sensitive information. They can be characterized by employing either no, software assisted, or hardware assisted protection.

Table 4.2 compares these types of key management systems by the kind of attack they are protected against. It lists these six types of systems and shows if they provide protection against an attacker with either logical access to the client machine, limited physical access, or full physical access. Full physical access assumes an attacker is able to modify the system in a way to gain access to the hardware protected secure storage holding encryption keys or authentication credentials. Limited physical access assumes that the hardware protection can't be circumvented.

The machine accessed by the attacker is assumed to be the *client* in the client-server configuration, and the protection method applies to the *authentication credentials* the client hold for authenticating against the server.

If an attacker gains access to the key server, rows one through three (local key storage) apply to the security of the key server content. Since a physical separation is in place in the client-server configuration, a successful attack is comprised of a successful attack against both the server *and* the client.

Descriptions of abbreviations in Table 4.2

LK Local key storage

KS Central key server

No No protection for locally stored sensitive information

SW Software (operating system) assisted protection

HW Hardware assisted protection

Key management		Protects against			
Type	Protection	Logical access	Limited physical access	Full physical access	Key revocation
LK	No				
	SW	✓			
	HW	✓	✓		
KS	No				✓
	SW	✓	✓	✓	✓
	HW	✓	✓	✓	✓

Table 4.2: Comparison of Key Management Schemes for Encryption Aware Clouds

The different relationship between encrypted data and the stored secret in local and centralized storage allows for a higher security standard and flexible management when using a key server. While locally stored *encryption keys* are directly related to the encrypted data and allow decryption without any further information, the stored secret on the client system using a key server is only connected to the encrypted data through the server. This allows the remote revocation of client credentials in the case of a physical attack which is detected through the interrupted operation of the system.

Local Key Store without Protection

Such a configuration does not provide any protection against an attacker with even only logical access to the system. Any encryption key can be read from storage just like the encrypted data itself. The configuration defies the purpose of encryption and is only mentioned for completeness.

Local Key Store with Software Assisted Protection

In this configuration, the encryption keys for the locally stored encrypted data are stored on the same hardware but are protected by software measures like file permissions. Given the software operates as intended, an attacker with logical access is not able to read the encryption keys. Physical access allows an attacker to circumvent these protection measures by replacing the software, or directly accessing the storage device.

Local Key Store with Hardware Assisted Protection

Instead of using software measures to protect the encryption keys, this configuration uses a hardware key store based on a *Trusted Platform Module* (see Section 4.3) that is capable of verifying the integrity of the operating system and software stack running on the machine before allowing access to the keys. As described above, it is assumed that an attacker with

full physical access is capable of circumventing this protection by gaining possession of the hardware and manipulating it with the adequate tools.

Client-Server Configuration without Client-Credential Protection

In this configuration, a client-server architecture is introduced and access to the stored content on the client doesn't allow the decryption of any data since the keys reside on the server. If the authentication credentials the client holds are not protected in any way, an attacker can forge a key request and send it to the server using the clients legitimate credentials. The server can't distiguish this request from a legitimate one, and the attacker gains access to the encryption keys.

Client-Server Configuration with Client-Credential Protection

A new functionality introduced with the client-server architecture is the possibility of revoking compromised client credentials. An advatage gained by this feature is the possibility to convert an attack *detection* mechanism to a *prevention* mechanism. This is done by monitoring the clients remotely and revoking the credentials on the server in case they become compromised.

Due to this functionality, even software assisted client-credential protection is sufficient to prevent any kind of attack.

4.3 Security Concerns

Encryption Scheme

As stated earlier, the contents of a key management system for encryption keys and system credentials comprise critical data that needs to be encrypted. An encryption scheme describes a method through which these keys are provided, and which entity holds them. In a distributed key management system, this can be either the client or the server. A distributed system has the advantage of physical separation between the encrypted data on the client and the keys on the server. This allows the clients to be put into a lower security environment while only a single server has to be protected in order to secure the whole system.

This model of having many low-security clients and a small number of protected servers allows easy deployment of such a system in existing data centers and creates the requirement for a flexible client management system on the server to introduce new client, modify access or remove them.

An encryption scheme putting encryption keys onto the *clients* defeats the purpose of such a distributed system since it introduces new requirements that conflicts with the ones initially motivating such a system.

The clients now contain key material which, together with a snapshot of the server's encrypted content, allows decryption of secret data on the clients. This raises the security level required for client instances.

Client management is more complicated in such a setup since the need to remove clients requires each client to have an individual encryption key which in turn has to be used to encrypt a new copy of each secret object on the server. Removing a client now requires to remove each secret object from the server that was encrypted with its key.

Keeping the encryption key for the secure storage on the *server* allows fulfilling the initial security requirements and provides a more flexible client management. The server uses a single encryption key for the secure storage and only has to manage access lists and client lists to enforce access restrictions.

Providing an Initial Master Key

In a setup that keeps the encryption key on the server, additional mechanisms have to be in place to provide separation between the secure storage content and the key used to encrypt it. This Master Key must be available to the key management server in normal operation but not to an attacker gaining access to the server.

Physical separation of the key and storage is achieved through a hardware key store which keeps the key in a separate tamper resistant memory module that is only accessible through protected channels.

Operating system protection provides another set of means to protect access to certain objects and enforce restrictions on the operation of the system. Compared to the physical separation achieved through a hardware key store, the restrictions enforced by the operating system are only in place as long the operating system is working. They are not tamper resistant and can be circumvented by exploiting vulnerabilities. This makes it necessary to restrict physical access to the server, and only expose those interfaces needed for key management operations.

Storing the Master Key

When using physical separation through a hardware key store or logical isolation through operating system means, the two stages of accessing the master key, and operating with it in a secure context have to be taken into account.

A **hardware key store** is a cryptographic, tamper resistant hardware module containing secure storage for key material and cryptographic processing capabilities. Often used in public key cryptography, the hardware module operates on the stored key material, providing capabilities to encrypt data, sign certificates or do authentication through a challenge-response protocol. Access to hardware key stores is possible through the PKCS11 protocol described in Section 4.4.

Hardware key stores provide security by storing sensitive key material in an external location that only allows access after successful authentication. When used to store the master key for the server, this authentication against the hardware key store has to be provided on server start up.

This can be done either manually, by credentials stored on the system or by using a hardware key store that authenticates the system itself through a *trusted Platform Module*. The TPM authenticates system hardware components and verifies the software before allowing access to the encrypted content only to the legitimate system it is bound to.

Using **local storage** to keep the master key has the advantage of not requiring additional hardware but provides weaker protection against attacks than a hardware key store with TPM protection. Anyone with physical access to the machine can read the plain text master key and gain access to the key management system content.

Preventing Access to the Master Key

Operating system methods can be employed to prevent unauthorized access to the stored master key. The basic scheme to such a setup is to first assign a dedicated process user ID to the key management server process, and then limiting access to the master key to that user ID. The general discretionary access control (DAC) mechanisms in UNIX operating systems don't allow for such fine-grained access controls. A root user would always be allowed to read the master key and can even delegate that right. Side channels for accessing the file content exist through reading the memory content of the key server process or directly accessing the storage device that holds it on a block level below the file system, circumventing file system access controls.

The SELinux kernel extensions provide such fine-grained mandatory access controls (MAC) capable of assigning access policies to files and system components that can't be delegated and even prohibit access by a root user [KAAS11]. To make use of these mechanisms for the key management server, a dedicated user account without login rights is created, and given sole access to the master key file. The user is then prohibited to run any process beside the key management server. On startup, the system automatically starts the key server under its dedicated user ID and assumes operation.

4.4 State of the Art Solutions

Public-Key Cryptography Standards (PKCS)

The Public-Key cryptography standards are a set of 15 standards covering generation of public key pairs and certificates, encryption algorithms and key exchange. Some of these standards will later find use in the design and implementation part of this thesis.

The PKCS12 standard defines an exchange format for cryptographic objects like keys and certificates. A PKCS12 key store is contained within a `.p12` file and provides encrypted storage for different objects. It is often used as a container format for storing a public key pair and an associated, signed certificate. The content can be secured individually by providing a passphrase for each security object, or as whole by providing a key store password. Each entry in a PKCS12 store is identified by an alias that has to be provided when storing, or retrieving content.

Libraries for operating with PKCS12 key stores are provided by the *Java Cryptographic Architecture (JCA)* through the PKCS12 key store implementation.

The PKCS11 standard [sta09b] defines the functional design and a common API for cryptographic hardware that provides hardware cryptographic functions, key storage and protection. As a common interface standard, it allow developers to write software that is capable of using hardware cryptographic modules without the knowledge of the specific module used by the customer.

Such modules are often used to contain a user's private key, and can be unlocked with a passphrase.

Websphere Keystore

The Websphere application server features an integrated key management solution that is used by Websphere itself to store its internal passwords but is also available to the application developer for storing credentials. Its intended use is in a secure environment where attackers are not supposed to gain access to the content of the store system containing the Websphere configuration files.

The key store is part of the `security.xml` configuration file which holds user names in plain text and passwords in an encoded form. Encoding is done by applying an `XOR` operation to the password and a cipher string, resulting in an obfuscated password. As mentioned in chapter 2, encryption mechanisms often rely on `XOR` operations in the last step of transforming plain text into cipher text by applying an `XOR` operation to the plain text and cipher stream produced by the encryption algorithm.

The encoding mechanism employed by Websphere does not make use of an encryption algorithm to produce the cipher stream but rather relies on providing a static cipher string that is sufficiently long to accommodate for the maximum allowed password length of 60 characters.

A known-plaintext-attack can always be used to reconstruct the cipher stream which is of minor concern in cryptographic applications since the cipher stream is unique and non-repetitive. When a static cipher string is used, its reconstruction by a known-plaintext-attack renders the whole encoding scheme futile. To make matters worse, the cipher string used by Websphere cannot be changed by the user and is shared across all Websphere installations. An attacker with access to any Websphere instance can therefor reconstruct the cipher string and decode any password file he gains access to.

This encoding mechanism shall serve as an example for an applied security-through-obscurity scheme whose stated purpose is not to secure the system against a dedicated attacker, but to deter the casual observer.[1] Its goal is to prevent a person getting a quick look at the configuration file from remembering the readable chosen password by transforming it into a series of hard to remember random characters.

To gain better security, Websphere provides a plug point to integrate a custom encoding mechanism for its key store that can utilize any encryption scheme chosen by the developer.[2]

Implementation

To use the Websphere key store for custom applications, a developer has to implement interface code for retrieving the credentials and provide them to the key store as part of the system configuration. The latter task can be done by either directly manipulating the `security.xml` configuration file or by utilizing Webspheres administration interface and entering the credentials as a new *Authentication Data Entry*.

Access to these entries is done by utilizing the *Java Authentication and Authorization Services (JAAS)* through a *JNDI* call that presents an interface to the stored credentials.

[1] https://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.express.doc/info/exp/ae/tsec_protplaintxt.html

[2] https://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.express.doc/info/exp/ae/tsec_plugpoint_custpass_encrypt.html

5 Cloud Solutions and Security

5.1 Threat Assessment and Standardization Efforts

As the importance of cloud computing in the private and official segment grows, governmental and non-governmental organizations are concerned with the security of these new service models. To combat the threats, these organizations work to raise the awareness of potential customers about the security implications and best practices while standards are being created to allow an objective assessment of the offered security.

In the following these publications will be analyzed and compared to pre-cloud security standards and recommendations that are not geared towards cloud computing but are still held in high regard among corporations and governments.

The cloud security alliance (CSA)[1] collects information on common threats towards cloud computing and issues a list of top threats concerning cloud computing itself, and its customers [CSA10]. The most severe issues pointed out can be traced back to an uncertain security standard and the lack of separation between customers. The latter constitutes the matter of this thesis and was found to be an issue of components and technologies not geared towards the use in cloud environments. The uncertainty of customers in the security of cloud environments can be traced back to the lack of common security standards for cloud computing, and the unsuitability of established standards for cloud environments.

A thorough survey conducted by the *European Network and Information Security Agency (ENISA)*[2] found a list of technology inherent threats not specific to cloud computing that must be considered when building cloud services [eni10]. It was found that due to the multi-tenant nature of cloud services, a lot of the attacks can be done by external attacks while they would require a malicious insider in conventional on-premise information systems which raises the risk of these threads in cloud environments. These threats concern all layers of the application stack from the user interface to the physical infrastructure and will be analyzed in the section below.

In their publication *Security Recommendations for Cloud Computing Providers* [bsi10] the *German Federal Office for Information Security (BSI)* compiled a list of recommendations concerning the infrastructural and operational security of cloud environments.

[1] https://cloudsecurityalliance.org
[2] https://www.enisa.europa.eu

As most important measures for securing cloud computing against external attackers and malicious insiders a clear separation of duties and access permissions between the service provider and the customer was recommended as well as protective separations between customers.

Established standards are often not geared towards cloud environments and don't take into account the concept of multi-tenancy by either neglecting it, or hindering its efficient implementation by placing strict requirements not suitable for cloud computing.

The *IBM Information Security Standard ITCS104* [sta09a] which applies only to IBM-internal IT systems makes recommendations for the secure configuration of IBM products and the separation of customers. It does not allow for physical resources to be used by both internal, and external processes while multiple external customers may share physical resources. In the ITCS104 terminology, different customers are considered to be separate *zones*. Following the guideline, different zones may only be handled by physically separate hardware or through virtualization by an approved hypervisor like *PR/SM on System z* and *PowerVM*, making the deployment of multi-tenancy cloud offerings on anything higher than the hypervisor level impossible. Following the reasoning about the economy of scale from Chapter 1, such limitations don't allow for flexible and cheap large scale cloud offerings.

5.2 Tenant Isolation and Cloud Delivery Models

Since cloud computing is not a new technology on its own but rather a new way of using and configuring existing technologies, it inherits security threats from the technologies it employs and exposes them in a new environment, enabling attack vectors that were not possible before. [Sav11]

The different cloud delivery and cloud service models (see Section 1.1) combined with the different approaches to tenant isolation (see Figure 1.1) create a wide variety of possible cloud service configurations, each requiring different security measures.

Each layer in the application stack of a cloud service consists of software that provides possible vulnerabilities that an attacker can take advantage of. Depending on characteristics of the cloud model and tenant separation, these vulnerabilities have different levels of exposure.

The cloud service model describes which level of the application stack is directly accessible to the customer, while the tenant isolation model describes which of the layers below that are shared among multiple tenants. A cross-tenant attacker has to penetrate all tenant-separated layers until he reaches a shared layer that allows access to another tenant's data.

Table 5.1 shows the possible combinations of tenant isolation and service models with their respective attack path, concluding that high level services with low level isolation provide the best security through a long attack path. With low-level cloud services like IaaS and PaaS, only few layers of tenant separation are possible, providing short attack paths by definition.

Isolation	SaaS	PaaS	IaaS
Application level	Enter: UI Target: Application	n/a	n/a
Application instance	Enter: UI Target: Middleware	n/a	n/a
Middleware instance	Enter: UI Target: Platform	Enter: Middleware Target: Platform	n/a
Virtual machine	Enter: UI Target: Hypervisor	Enter: Middleware Target: Hypervisor	Enter: Platform Target: Hypervisor

Table 5.1: Tenant Isolation and Cloud Service Model Combinations

5.3 Common Threats to Cloud Components

An attacker with a subset of the capabilities of an idealized adversary can be a member of either of three groups with different preexisting access to the cloud application. An **external attacker** has no legitimate access to the application and may only be able to reach an initial login page or even be shut out by a VPN. A **cross-tenant attacker** uses his legitimate access to the application as a means of accessing other customer's data. A **malicious insider** is an administrator or other person with legitimate physical or virtual access to some service within the application who uses this privilege to gain access to customer data.

The cross-tenant attack is an attack vector available to attackers with legitimate access to the cloud application as a customer. This access is used as a gateway to the system and ultimately other customer's data.

A cross-tenant attacker has a superset of the capabilities of an external attacker, while gaining these capabilities is often as simple as signing up for the cloud service. These circumstances have made the cross-tenant attack an attractive attack vector [DMT11].

A typical SaaS cloud computing solution is comprised of a software stack containing an operating system on virtualized hardware, middleware components like a database or application server, and an instance of the cloud application on the server side. The client side executes the user interface through a web browser.

In order to secure the system against cross-tenant and external attacks, the tenant separating layers must only allow requests from legitimate user operations to pass through to the tenant-shared layer to access data. In order to prevent an attacker from penetrating a layer, the software itself must not contain any known vulnerabilities that can easily be exploited. Once this requirement is met, the software must be used in a secure manner by utilizing the security features it provides through setup and configuration procedures. This includes custom made components, off-the-shelve middleware components as well as the operating system and hypervisor.

In order to compile a set of recommendations for securing cloud components, this section will analyze actual and researched attacks on software and hardware components in the context of cloud computing and other applications, and discuss methods to avoid the underlying vulnerabilities. Table 5.2 gives an overview of the vulnerabilities and their parameters discussed in the following Sections. It lists the type of attack, abbreviations used for the attack types are found in the detailed description in the following Sections, the point of entry an attacker has to use and the according access an attacker needs to the system. The popularity estimation is given according to the findings in the IBM X-Force reports [XF10] and [XF11].

Attack Type	Point of entry	Required access	Attack target	Popularity
XSS	User interface	UI access or CSRF payload	User interface	High
CSRF	User interface	None	Service layer	Medium
SQL-injection	User interface	UI access or CSRF payload	Service Layer	High
MITM	Network infrastructure	Network component	Communication	Low
VM-escape	Operating system	Full OS access	Hypervisor	Low
OS-escape	OS process	Access to process	Operating system	Low

Table 5.2: List of Threats and Vulnerabilities in Cloud Computing

Custom Web Applications and User Interface

Cloud services are built using off-the-shelve components on the infrastructure and middleware layer that are customized through configuration, and custom made components that comprise the service layer and user interface.

The *2010 IBM X-Force® Trend and Risk Report [XF10]* found that 49 percent of all vulnerabilities discovered in 2010 were found in web applications. This is attributed to the fact that these applications have a shorter life cycle than existing middleware of infrastructure components and therefore underwent less testing in average. An important factor for the finding of these vulnerabilities and the risk they pose is their level of exposure. They comprise the topmost layer in the software stack and are directly accessible to an attacker.

Code Injection Vulnerability

The SQL injection attack as well as XSS discussed below are special cases exploiting a code injection vulnerability. Code injection in general allows an attacker to have custom code executed by, and in the context of the system under attack.

Systems that allow the ingestion of files or data often operate on that data to extract information or perform transformations. This user provided data can be used by an attacker as a means of injecting code into the system that is executed by exploiting a vulnerability in the component operating on the data. This attack provides a method of exploiting vulnerabilities in hidden software components.

Code injection is used to penetrate the layer the attacker has access to, and either execute the attack on the layer below, or penetrate the stack further by exploiting vulnerabilities that are now accessible.

Cross-Site Scripting Attack (XSS)

The cross-site scripting attack uses a vulnerability similar to the SQL injection attack in that it relies on user input not being checked or sanitized. Contrary to the SQL injection attack, XSS does not seek to alter the behavior of the service layer, but rather the user interface. It works by injecting code disguised as user input into the application and have it become part of the user interface logic. [DLFMT04]

The goal of this attack is to alter the behavior of the user interface in order to steal information being displayed in the user interface, or entered by the user. Compared to the SQL injection attack, the attacker has the advantage of full knowledge over the user interface code since the JavaScript and HTML that makes up the UI is interpreted by the browser and thus has to be available in plain text to the client.

The attacker generates JavaScript code that replaces or expands the functionality of the service-provided user interface code and injects it into the application by exploiting a vulnerability. He can redirect any information present in the user interface to a remote server, or trigger operations on behalf of the user to his benefit. XSS is often used to steal HTTP-cookies containing session tokens from a user in order to hijack the session. This attack circumvents a security measure in web browsers that prevents them from sending cookies to a different web-domain than the one they were received from. Since the injected script runs within the context of the legitimate web application, it has full access to the HTTP-cookie belonging to the user session and can send it to the attacker[3].

Attack Vector Two types of XSS attacks can be identified. Persistent and reflected XSS. A persistent attack uses a vulnerability that allows storing the XSS code in the persistence layer of the application, and having it repeatedly delivered to users in the future. A reflected attack manages to inject code into a a single instance of a user session.

Figure 5.1 shows the attack vector for a persistent XSS attack where the attacker and user access the same application with some shared data. The attacker places the XSS code in the shared content that is later retrieved by the user's browser

[3] http://www.cgisecurity.com/xss-faq.html

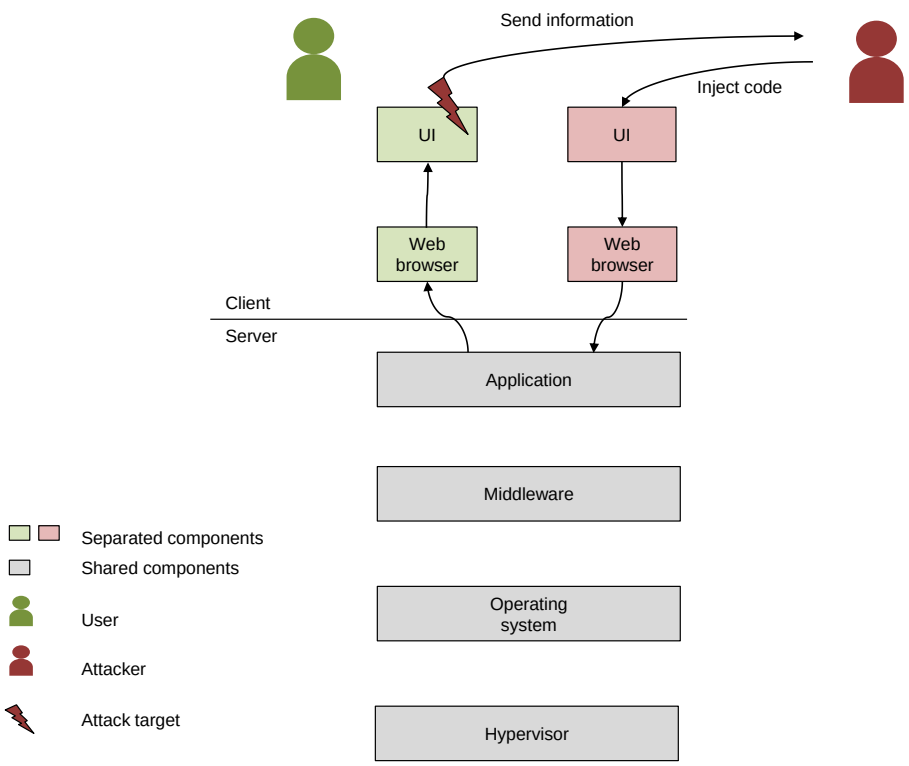

Figure 5.1: Attack Vectors: Cross Site Scripting

The usual point of entry for reflected attacks are URLs. Dynamic web applications often use URL parameters to pass values on from one page to the next. These parameters are then used by the target page to include content or determine functionality. If content provided through a URL parameter is included in the resulting page, an attacker can prepare a malicious URL linking to the legitimate target application, with the attack code hidden as a URL parameter. He then lures the target user into clicking the malicious URL. The user is presented with the application while his browser executes the script provided by the attacker.

Persistent XSS attacks exploit any functionality of the application that allows a user to store data that gets retrieved and displayed in the user interface. This type of XSS is more versatile than the one-time attack since it allows targeting a larger user group without raising suspicion through manipulated links.

Mitigation XSS attacks are prevented by sanitizing user input from JavaScript code, and treating any external data as user input.

Cross-Site Request Forgery Attack (CSRF)

Contrary to the XSS attack that relies on the trust the web browser has towards the server delivering the infected content, CSRF preys on the trust the web server has in the client's browser. The goal of the attack is to trigger an operation on the server on the user's behalf that is beneficial to the attacker.

The attack requires a user to be logged in at the targeted service, and have him click a URL on a malicious site that points to the targeted service and triggers an operation as a side effect.

Figure 5.2 shows the attack vector for a CSRF attack.

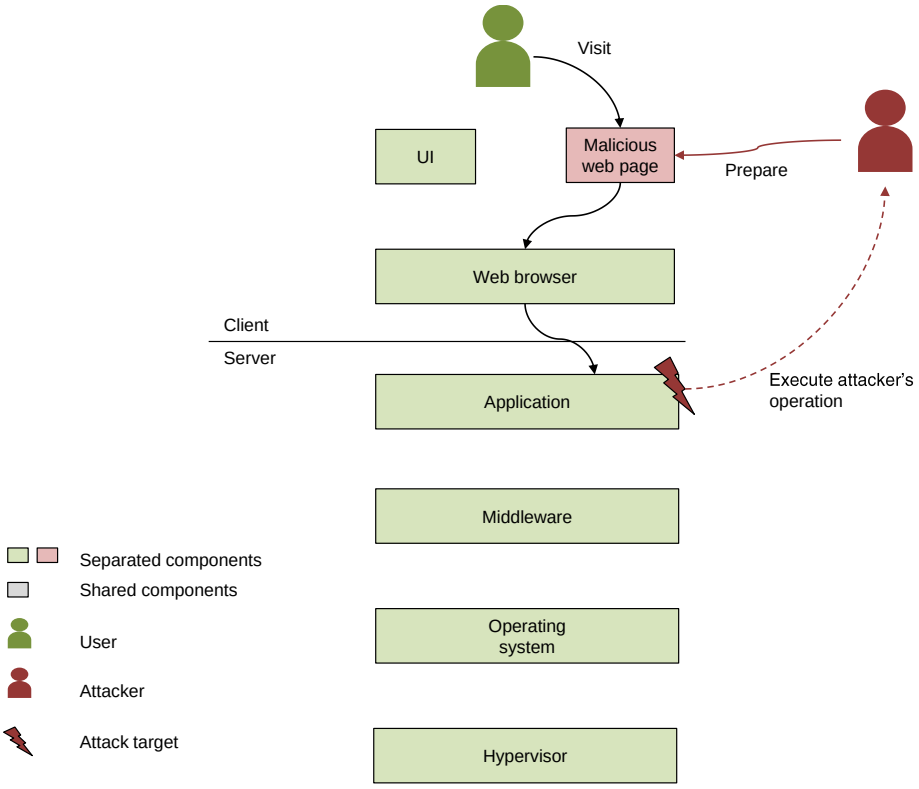

Figure 5.2: Attack Vectors: Cross Site Request Forgery

This attack exploits a design aspect of HTTP-cookie handling in web browsers. A server can issue a session cookie that is saved by the browser and later sent back with every request to the server. This mechanism allows the web server to keep a user authenticated after the initial log in, and identify him on every successive request.

Attack Vector Attackers exploit this behavior of browsers by sending a request to the server from within the browser through a prepared web page that they trick the user into visiting. The browser dutifully includes the HTTP-cookie with the request, and the server assumes it is a legitimate request.

Assume a banking application had a method for transferring funds that is triggered by calling the URL `http://bank/transferFunds?from=victim&to=attacker&amount=1000`. In a legitimate use case, this URL would be called by a page after the user entered the transfer parameters. An attacker prepares such a URL and places it on the site he lures the user into visiting. After clicking the URL the server recognizes that the users web browser is logged in, and carries out the transfer without the user noticing.

Mitigation The CSRF attack can be prevented on the client side by browser security extensions, or on the server side. In a large-scale distributed application like cloud computing, controlling the client environment is not feasible so that a server-side approach should be favored. [SV11]

The HTTP-header of the request sent by the client's web browser contains a `Referrer` attribute indicating from which page a request originates. Legitimate requests all originate from the user interface itself, while the malicious request originates form the attackers prepared web page. The server has to inspect the `Referrer` attribute and compare it to the URL of the legitimate user interface to identify and deny malicious requests.

However, `Referrer` validation has its limitations since the attribute (URL of the web UI) is not secret and can be included in the request if an attacker manages to modify the HTTP-header through a browser vulnerability.

A more secure approach found in the *Django-framework*[4] is to require a secret token with each request. Similar to the HTTP-cookie, this token is assigned to a user session and included in the requests to the server in order to authenticate the user. While the HTTP-cookie is managed by the browser and included in every request to the server independent of origin, the secret token is managed by the UI itself and is only included in requests generated by the UI.

The Prepared Web Site

As discussed before, reflected XSS and CSRF attacks require the user to visit a malicious web site which starts the attack by executing code within the context of the user's browser. The most effective method for an attacker is to abuse trusted, legitimate web sites that the user might visit. It has been shown that small set of software with well-known vulnerabilities runs a large portion of public web sites which provide an attacker with an easy way to plant his malicious code[5].

[4] https://docs.djangoproject.com/en/dev/ref/contrib/csrf/
[5] https://www.stopbadware.org/pdfs/compromised-websites-an-owners-perspective.pdf

Authentication Vulnerabilities

Vulnerabilities in the user authentication process include password security and the authentication process itself. Password security has been thoroughly researched in the past and a set of rules can be found in the IBM ITCS 104 guideline [sta09a] and other standards mentioned in Section 5.1.

The authentication process itself relies on the security of the involved components like the authentication provider and the security of the transmitted data which were analyzed in Chapter 3.

The security requirements for the mentioned authentication determine which of the attack vectors presented in this Chapter pose a threat to the system. The source-to-sink model used below to describe the information and authentication flow in SmartCloud Archive indicates which station in the authentication process is potentially vulnerable, and which is inherently secure due to the use of non-sensitive data. These characteristics of the authentication method must be reconciled with security measures for the involved components in order to prevent attacks.

For example, password based user authentication methods have an inherently vulnerable source since the user provided password can be repeatedly used for authentication even when stolen by an attacker. On the contrary an authentication provider for such a mechanism might only store secure hash values of the passwords which don't comprise sensitive information since they can't be user for authentication and therefore don't compromise the security of the system when disclosed.

Middleware Components and Service Layer

Access to middleware components is the business model for PaaS providers and therefore these components comprise the first line of defense against an attacker. In the context of SaaS applications, the middleware components are only accessible to an attacker through code injection vulnerabilities, or to a malicious insider.

While the XSS code injection attack discussed above is targeted at the user interface running inside the victims web browser, other code injection attacks merely use the user interface as point of entry, but are targeted at middleware components or the Service Layer.

SQl Injection Attack

The most common attack against web services is the SQl injection attack. It seeks to inject malicious SQL statements through the web service into the underlying SQL Database in order to extract valuable information, or manipulate the behavior of the application. The *IBM X-Force 2011 Trend and Risk Report [XF11]* found that the SQL attack is the most popular attack against the service layer of an application amongst attackers, making such a vulnerability a

high-risk item. Its popularity is attributed to the low access requirements an attacker has to meet in order to abuse such a vulnerability, and the high impact posed by it.

It uses a common vulnerability in web services that internally compile SQL queries that include user-submitted data. The attacker forms the string he submits as a user in such a way that it modifies the semantics of the SQL query compiled by the application to make it execute a different operation. This usually involves special characters that are used inside SQL queries as string delimiters. The attacker includes such characters in his string to escape the query syntax compiled by the application and have parts of his string executed by the database as SQL commands.

Due to the possibility of nested sub-queries, an attacker can include arbitrary commands in his attack and is only limited by his knowledge of the underlying database schema. The databases response may not be visible to the attacker depending on the applications interpretation of it, and the visual representation in the user interface.

The goal of this attack is to extract information from the database that is not intended to be visible to a user, or to modify information in order to alter the behavior of the application. In cloud configuration that has no tenant separation on the database layer, an SQL injection attacker has immediate cross-tenant data access. Figure 5.3 shows the attack vector for SQL injection. The separation of the layers above the database instance can be of any type. SQL injection can be used by a malicious insider to access data hidden from him, or by a completely external attacker in a cross-tenant attack.

Points of Entry This attack can be used against any component of the system that accepts user input and uses it in an SQL query like user submitted forms, login pages or URL parameters.

Mitigation The SQL injection attack is widely known today and the means to avoid it are present in most frameworks and programming languages. The approach is to sanitize user input before including it in SQL queries. Either by removing special characters, or transforming them into escape sequences that are not evaluated by the SQL query parser but retain the information of the special characters[6].

Another secure approach for preventing SQL injection is to use the SQL server itself rather than the application to sanitize user input. This can be done by utilizing SQL stored procedures and pass user input as parameters to these predefined procedures rather than building custom SQL queries in the application. The SQL parser will always use the parameter values only in the intended place and prevent the *parameter escape* approach used for SQL injection.

[6] http://www.unixwiz.net/techtips/sql-injection.html

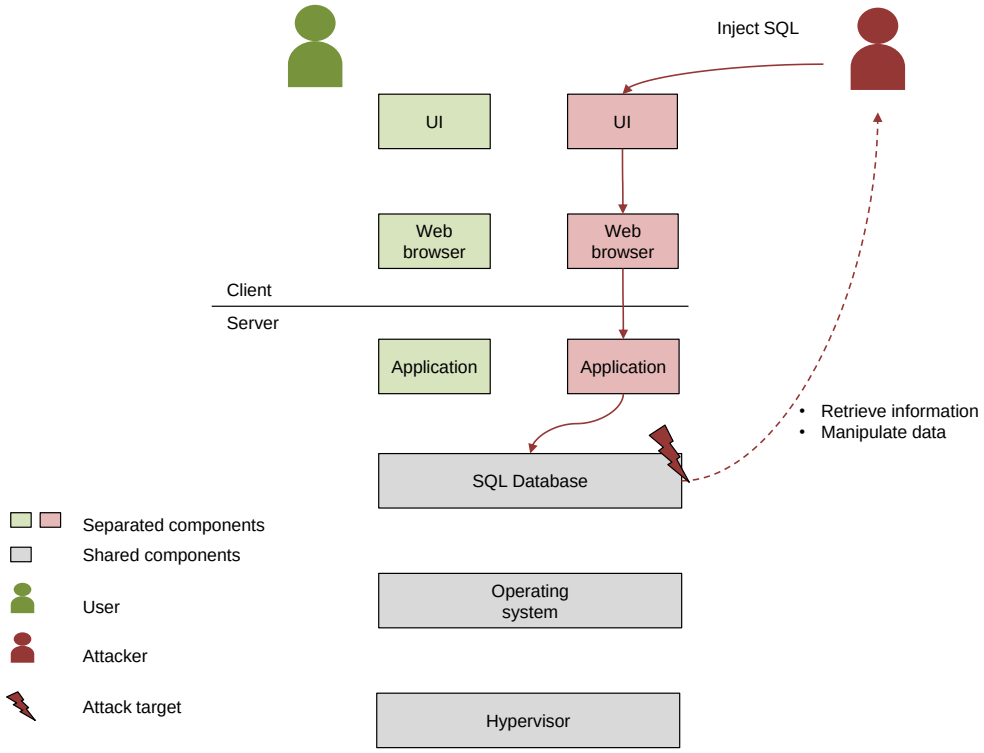

Figure 5.3: Attack Vectors: SQL Injection

Network Infrastructure

As a central part of the infrastructure of any cloud environment, the network layer carries any kind of information including sensitive data. Direct access to network infrastructure components is usually not possible for an external attacker, but often trivial for a malicious insider. Due to the nature of network transmissions, two basic kinds of attacks can be identified; unauthorized data access or eavesdropping, and data manipulation. The combination of these two consitiutes the *man-in-the middle attack* which seeks to intercept information, manipulate and resend the information. Figure 5.4 shows this attack vector. The goal of man-in-the-middle attacks is usually to prevent the detection of an attack by manipulating an operation without changing the outcome the user or system expects.

The term man-in-the-middle is often used ambiguously for all three types of attacks stated above, and usually assumes the attack not being detected.

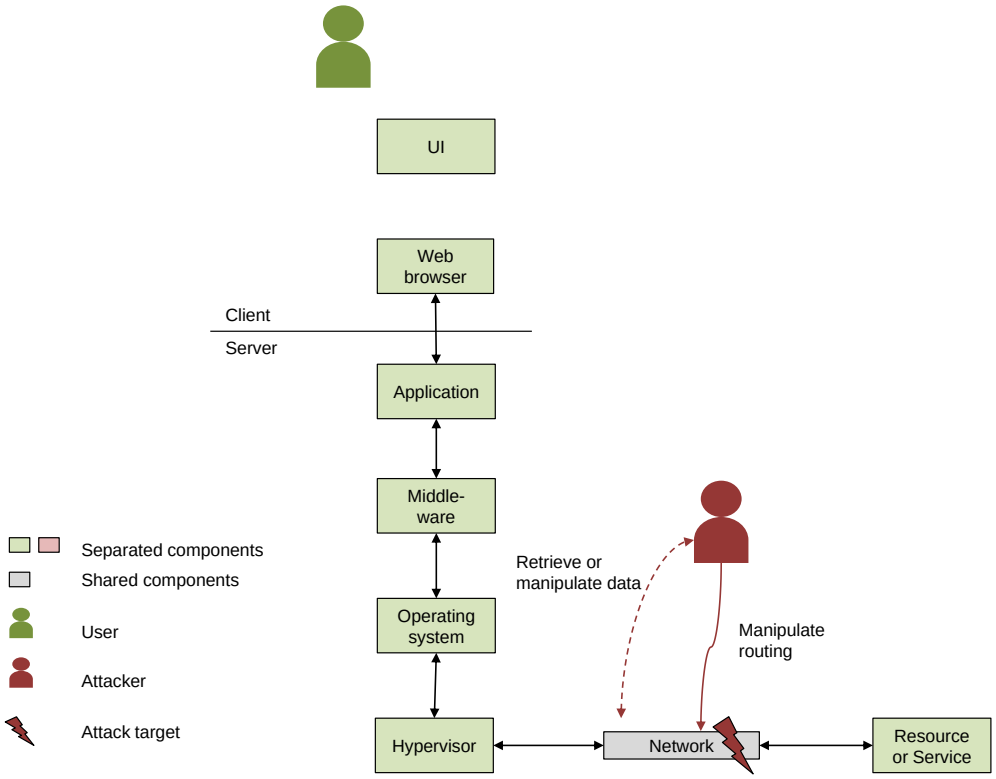

Figure 5.4: Attack Vectors: Man-in-the-Middle

Points of Entry

Any part of the network infrastructure between the source and the sink of a request be the point of entry for a man-in-the-middle attack. Misconfigured network devices or services are usually targeted in order to divert the original traffic to the attacker, and back. These can be hypervisors providing a virtual network, or hardware network appliances like routers or switches.

Man-in-the-middle originates from physically separating a wired network connection and inserting a custom device that would manipulate and forward the data traffic. A popular approach to logically diverting network traffic is found in the method of *ARP spoofing*. The *Address Resolution Protocol (ARP)* operates on layer 2 of the OSI model[7] and is responsible for resolving IP addresses to the physical (MAC) address of a network device. Any new network device introduced to the infrastructure advertises its IP and MAC address in order

[7] https://en.wikipedia.org/wiki/OSI_model#Layer_2:_data_link_layer

for routing devices to compile a table of logical IP addresses and their physical location on the network.

In order to divert traffic, an attacker will advertise his own physical address with the logical address of the target, and have all traffic to the target diverted to him.

Mitigation

The physical manipulation of network lines can only be prevented by physical means like access restrictions and surveillance since eavesdropping on a physical line is impossible to detect from the point of view of the network endpoints. It has been shown that without physically manipulating the line, an attacker can pick up the electromagnetic field surrounding the wire with a sensitive antenna and extract information about the data being transmitted, which rules out any possibility of detecting such an attack without physically monitoring the line.

ARP spoofing attempts are prevented by modern network equipment, and can be prevented by a secure network infrastructure configuration.

The common approach to prevent such attacks without reliance on a secure network is to use an encrypted communication channel. methods like public key cryptography discussed in Chapter 2 allow the establishment of a secure communication channel over an insecure network connection. Such an approach is based on the *detection* of an attack, and uses this knowledge to elevate the detection mechanism to a *prevention* measure by rejecting any information that was identified as part of an attack.

Operating Systems and Processes

The operating system constitutes the last layer of tenant separation in PaaS and SaaS offerings that don't use separate virtual machine instances per tenant. Cloud providers either strive for low service costs through a high degree of tenant integration, or a high level of security through a low level of tenant integration. Therefore separation on the operating system layer is usually chosen by PaaS providers since it is the highest level of integration possible in such a configuration.

This concept achieves tenant separation by using operating system mechanisms like processes and access control lists in order to prevent a tenant from accessing foreign data. Attackers in PaaS services attempt to penetrate this separation layer in order to access or manipulate data belonging to other tenants, while attackers in a lower integrated configuration with hypervisor separation attempt to break the operating system isolation in order to attack the underlying hypervisor. Figure 5.5 shows the attack path for breaking operating system isolation.

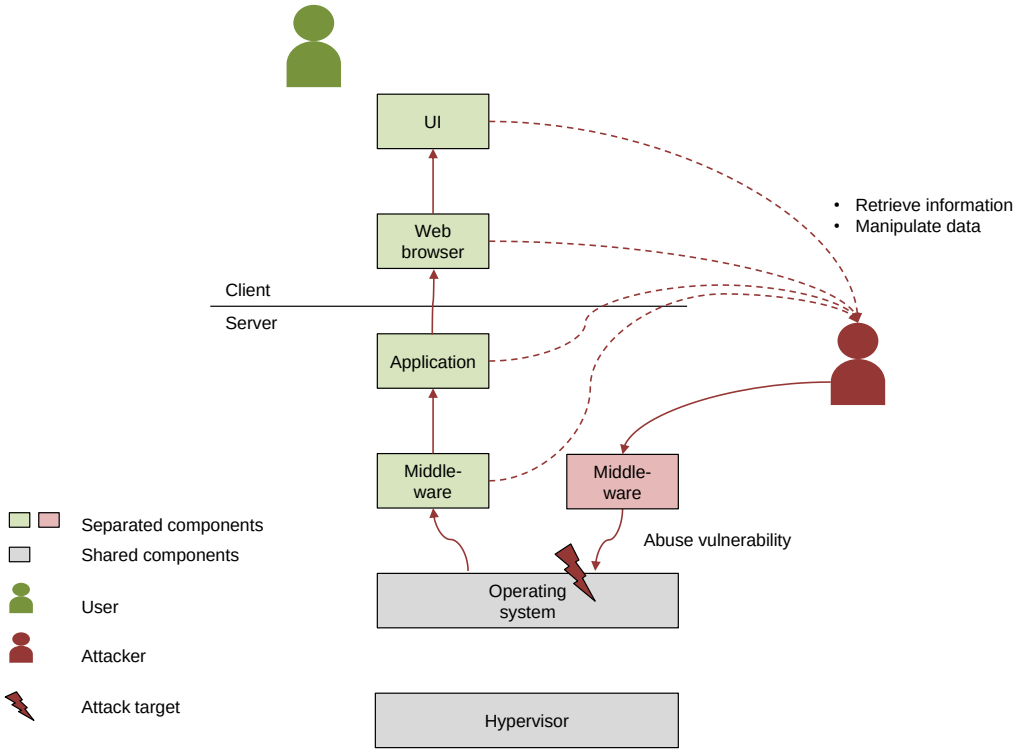

Figure 5.5: Attack Vectors: Escape operating System Isolation

Points of Entry

The rationale of operating system separation rests on the concept of a process context which consists of memory and access to other system resources. The most popular approach for attackers is to escape this process isolation in order to access memory belonging to another process, or the operating system kernel. This attack is considered a *privilege escalation* since the attacker seeks to gain higher privileges on the system than his process was originally assigned. This is done by either exploiting vulnerabilities in the operating system itself, or other processes that have higher privileges and provide some kind of service to the lower privileged service.

Since most PaaS and SaaS offerings consist of multiple services that are often shared between tenants or run with higher privileges than the services belonging to a specific tenant, a popular approach is to abuse vulnerabilities in these services to execute operations in their context,

Mitigation

A secure configuration of the operating system process isolation mechanism is key to operating system assisted tenant isolation. When such a configuration is in place, an attacker can only rely on vulnerabilities in either the operating system or the services. In order to mitigate these risks, the principle of least privilege must be applied and services should always run with the minimal amount of system privileges they require to work.

It has been shown that the abuse of vulnerabilities is often only possible through advanced execution concepts like run-time code generation that modern software often relies on [HAF+07]. Without these possibilities, the risk of abuse can me minimized while some advanced functionality is lost. Such functionality is present in all major operating systems today, but hardened software can be written by not making use of such features.

Virtualization Technologies

Virtualization constitutes the lowest possible layer in a multi-tenancy configuration, and the only layer of tenant separation in IaaS offerings. As the last line of defense, secure virtualization is a key requirement for secure multi-tenant solutions.

Prerequisite for an attack on virtualization technologies is an attacker with access to the virtualized operating system. This is the case in either an IaaS environment or in PaaS/SaaS offering where an attacker managed to circumvent the security measures in the abstraction layers separating him from the operating system.

The *IBM X-Force Trend and Risk Report 2010 [XF10]* has shown that a total of 37 percent of the vulnerabilities found in virtualization technologies allowed an attacker to escape the virtual machine and gain direct access to the underlying hardware, enabling a cross-tenant attack.

Other types of attacks against virtualization target either the software on the administrative VM or the guest VM images. The administrative VM is used to administer the guest VM instances and allows for the manipulation of network traffic and file access. Attacks on the guest VM images have the goal of modifying their behavior by manipulating the operating system or software code either while the VM is being transmitted over the network, or stored.

Points of Entry

Three kinds of virtualization related attack vectors were identified. The hypervisor-escape attack, VM manipulation during transport or storage, and the attack on the administrative VM.

Hypervisor-escape constitutes the most severe threat to virtualization environments today due to the low access requirements an attacker has to meet, and the severity of the possible consequences. The point of entry for such an attack is the virtualized hardware under the

operating system. Figure 5.6 shows this attack path. An attacker has direct access to these resources in IaaS offerings, and can gain such access by penetrating the abstraction layers in PaaS and SaaS.

Attacks on the administrative VM are usually carried out as payload for other types of attacks like viruses or malware. They seek to infect the software on the administrative VM in order to control it. This attack vector is not specific to virtualization.

Manipulation of the VM code is comparable to the manipulation of program code in conventional environments through viruses or malware. Points of entry for this type of attack are either a direct attack on a running VM instance, or manipulation during transport through the aforementioned man-in-the-middle attack, or during storage through vulnerabilities in other parts of the infrastructure that allow an attacker access to the storage system.

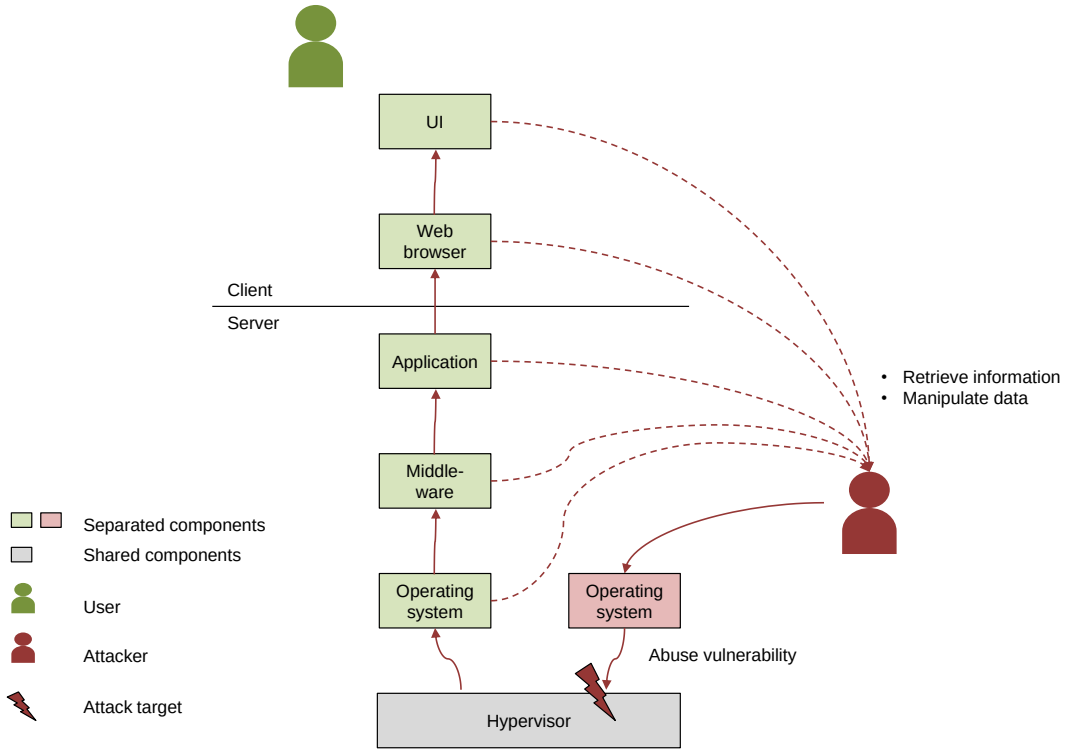

Figure 5.6: Attack Vectors: Hypervisor Escape

Mitigation

In [CSS+09] the technique of VM introspection was researched to help mitigate threats to the hypervisor that originate from within a guest VM. Since the hypervisor has full access to all resources used by guest VMs, hypervisor assisted VM introspection allows to analyze the processes running in the guest VM and detect attack patterns. No such technologies are currently available in commercial products, but they present a promising mechanism for securing virtualization environments.

Hypervisor escape attacks always work by exploiting vulnerabilities in the hypervisor and thus can only be prevented by using a secure hypervisor that doesn't expose known vulnerabilities.

Attacks on the administrative VM are prevented by limiting access to the administrative VM and securing it by the same means put in place for application servers and other critical infrastructure. They should be used for the administrative purpose and not have any unnecessary software installed or potentially vulnerable services running.

To prevent manipulation of the guest VM code, It should only be transmitted over the network in an encrypted form to prevent man-in-the-middle attacks, and must only be stored in way that allows the detection of manipulation.

Physical Security

All attack vectors discussed before are possible with logical access to the system and can therefore be perpetrated by any attacker external, or internal. When all vulnerabilities allowing malicious logical access are closed, the only threat remaining is physical access.

Physical access to the system is possible for *malicious insiders* and the prevention of such attacks requires physical measures as well. Logical access to the operating system layer as described above can be deterred through a secure configuration of the operating system while physical access allows an attacker to circumvent these safeguards. The threat of malicious insiders is faced by any information system but is particularly severe in cloud computing due to the high density of customers in a data center and the uncertainty of the physical location, as well as the personnel handling it. The surveys [CSA10] and [eni10] mentioned in Section 5.1 both point out the malicious insider as one of the major threats in cloud computing for these reasons.

Sensitive information residing on the system can only be protected from a malicious insider by physically preventing access. To make this solution feasible, data encryption as described in Chapter 2 is used to reduce the amount of sensitive information down to a small number of encryption keys that can be kept physically secure with reasonable effort. In such an environment, data security relies on the security of encryption keys and the anatomy of a key management system must be designed in such a way that legitimate operations can acquire keys while unauthorized access must be denied.

Attack Vector

The goal of the malicious insider is to abuse his physical access to the infrastructure to gain access to the sensitive data by either accessing unencrypted data, or in the case of encryption, acquire the data and respective encryption key.

The physical attack against a system is comprised of either replacing the operating system and software in order to circumvent any software-based safeguards, or removing hardware components in order to extract data using the attacker's equipment.

Mitigation

As shown in chapter 2, an encryption aware cloud manages and uses encryption keys to secure content in storage and transmission. Chapter 4 discussed different methods of storing and providing these keys within the cloud and has shown the security concerns with storing encryption keys on the same physical machine as the encrypted content.

5.4 IBM SmartCloud Archive — Architecture and Components

As an enterprise cloud based SaaS solution, IBM SmartCloud Archive (SCA) is comprised of an architecture of existing off-the-shelf, and custom made components. The security of such a system depends on vulnerabilities in custom made components and on the interaction between the components, assuming the external components are safe. These two topics are left to evaluate.

- Internal application security
- Communication and component interaction

The second item relies on leveraging secure communication channels and the security of the external components. These components are assumed safe in the sense that the developer and system architect using them has no means of evaluating the security of their implementation and design. External components offer a set of security related features and configuration parameters that when used allow the architect to integrate them securely into a compound system like SCA. The components that make up SCA and its architecture are described in Figure 5.7.

IBM Websphere® Application Server

The middleware component Websphere Application Server runs all the components that make up the SCA service and provides methods to allow communication between these components. This makes it the key part for securing the whole system. Through JDBC it acts as a database connectivity provider for its applications and offers JAAS to provide

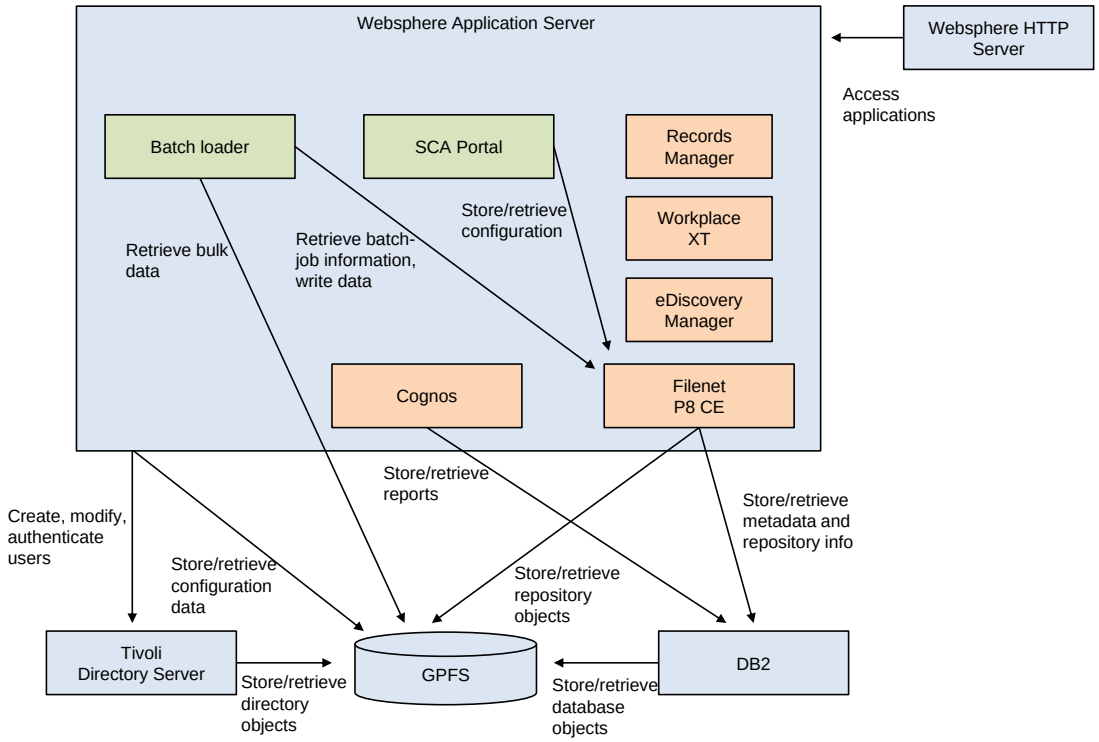

Figure 5.7: IBM SmartCloud Archive component relationship

user authentication against the IBM Tivoli Directory Server. Users access the system via the Websphere HTTP Server which serves the user interface components and redirects requests to the applications within the Websphere Application Server.

IBM Filenet P8

IBM Filenet P8 is an extensible enterprise content management system and serves as the central repository for customer data within SCA. It follows the object-store model and allows storing documents and data as objects with attributes, meta data and relationships. The repository content is stored on a file system while repository management information is kept in a database. The Filenet repository can be extended with add-ons to provide further functionality. SCA makes use of the Filenet Records Manager, Workplace XT, eDiscovery Manager and Content Search Engine. The Filenet Records Manager allows documents to be declared as (legal) records which have a certain retention policy that dictates a time frame in which the document may not be altered or deleted since it must be available for legal purposes. Workplace XT is the front end application for the Filenet repository which is used by customers to add and manage the repository content. Legal cases require certain

documents and records relevant to the case to be identified and retrieved from the repository. This task is done by the eDiscovery Manager and Content Search Engine.

The Filenet repository is accessed by its add-ons through user-initiated operations, by the SCA Batch Loader through scheduled tasks and through processes within the SCA Portal application. In order to secure access to the repository content, Filenet utilizes Access Control Lists (ACL described in Section 3.2) and user authentication.

IBM SmartCloud Archive Portal and Batch Loader

Since IBM already has all the components to create an archiving solution available, only few parts of the SCA system have to be developed from scratch.

The SCA Portal provides a central interface for configuring and managing the system and allows the customer to use the solution with little knowledge of the underlying components. It is used by a small group of customer employees to perform configuration and management tasks and is not available to the wider user base.

Operations in the SCA portal are triggered by either an authenticated user or a scheduled system task. Access restrictions are provided by Role Based Access Control (RBAC described in Section 3.2) with a set of predefined roles that can be granted to system users. Interactive users are authenticated when they access the system. Scheduled system tasks are created by such users and need to authenticate themselves against other system components once they are started. This requires the user setting up a task to provide (his own) user credentials in the form of a username/password with sufficient permissions when configuring a task. These user credentials are then stored in plain text within the task definition, which the batch loader later reads to execute the task and use the contained user credentials to authenticate against other system components.

The SCA Batch Loader adds a second way for data to be imported into the Filenet repository besides Workplace XT. It was designed to ingest large amounts of data in scheduled batch jobs and allows the customer to keep the SCA repository in sync with their operational data stock.

IBM Tivoli® Directory Server

The IBM Tivoli Directory Server is used as a centralized LDAP user authentication service in the SCA architecture. Every user with access rights to the system is represented by an object within the directory tree of the LDAP server. The user object properties include a short name (`sn`) and a password (`userPassword`) to authenticate the user. User passwords are stored as a salted SHA hash value to eliminate the risk of unauthorized data access. To implement the Role Based Access Controls used in the SCA Portal, the LDAP directory contains a group object for each available role. Users are added to the groups corresponding to the roles they have been assigned to.

Access to the LDAP server is needed for generally two different kinds of operations; user authentication and user management. The use case for user authentication consists of the user providing his credentials and the LDAP server verifying these. Security requirements for this operation can be minimized by calculating the hash value of the password before sending it to the LDAP server so no critical data has to be transmitted.

User management includes creating, removing and modifying users as well as managing group membership. These operations have to be performed by a user with higher privileges on the LDAP server, and some of them require transmitting sensitive password data in plain text.

5.5 IBM SmartCloud Archive — Processes and Activities

After deriving basic security recommendations from the systems components and their functions, the system activities have to be inspected in order to identify the access patterns and requirements imposed by the system.

IBM SmartCloud Archive (SCA) combines a number of different IBM products and custom components into an integrated solution. Communication between these components needs to be secured by authentication and authorization in order to utilize the integrated security mechanisms of the different components. Following the *Principle of least Privilege* every system access should only be granted the absolutely necessary rights and privileges in order to perform the desired operation [Am094].

This requires the use of credentials to authenticate against system components and gain access to these operations. The propagation of credentials in chains of components and the management of privileges associated with the subject belonging to them will be discussed in the following, leading to one application for the proposed cloud key management system described in chapter 6.

Every operation performed within SCA has a **source** and a **sink** of credentials. The source triggers the operation which is then propagated through a set of SCA components until it is executed by the sink. In order for a secure system, the source needs to be authenticated and that authentication must be passed on to the intermediate components in order for the sink to verify the legitimacy of the requested operation.

A source of an operation can be one of the following:

- An authenticated subject (user)
- An internally triggered task

In the current SCA architecture an operation triggered by a subject begins with the subject's authentication and the assignment of a session token which is later used the sink and other components to identify the subject. Figure 5.8 shows this process.

Internally triggered tasks don't use the token mechanism but rather supply credentials directly to the components they need to authenticate against.

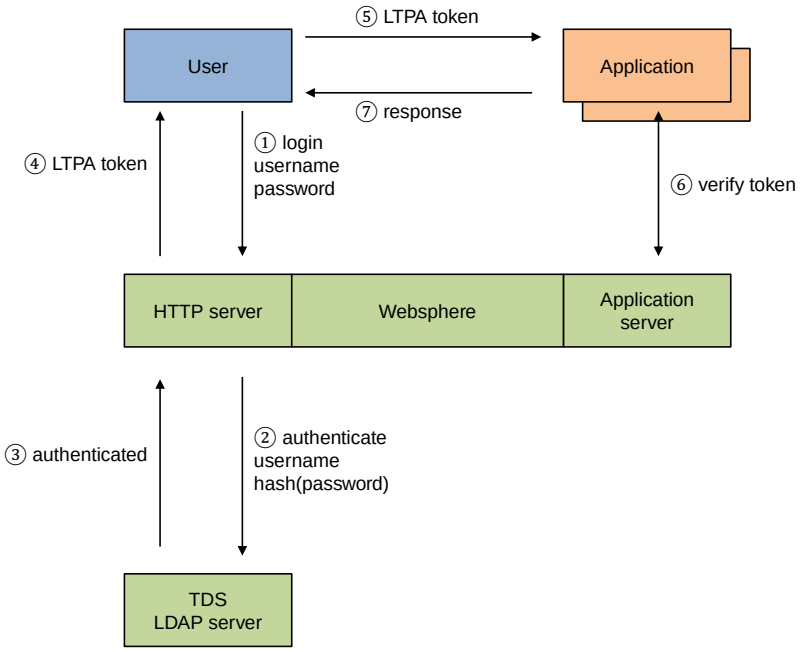

Figure 5.8: Authentication of a subject in IBM SmartCloud Archive

The sink of an operation can be:

- The DB2 Database for configuration changes or system task definition.
- The Filenet P8 content repository in the case of content insertion or retrieval.
- The Tivoli Directory Server is considered a sink when user credentials are modified or created.

For the proposed Cloud Key Store design, the internally triggered Batch Load process in SCA is modified to use CKS as a source of authentication credentials.

The Batch Load process involves to stages; fist the process is set up by a user or administrator, and later the scheduled task is executed by system using the stored credentials that were provided by the user or administrator in the first step. As stated above, the current implementation for this process stores these credentials as properties within the task definition, leading to the storage of sensitive information in a location not designed for this purpose.

As a storage and management system for sensitive key material and system credentials, CKS provides a method of storing this information in a secure location and only allowing authorized access.

Due to the design of the Batch Load process and the overall system architecture, the following requirements for the Cloud Key Store are necessary.

The work flow of setting up a Batch Task and executing it dictate these requirements:

Requirement 1. *During an interactive user session, system credentials must be stored in CKS.*

Since the principle of least privilege applies, reading or writing the credentials should only be possible if the task requires it.

Requirement 2. *The scheduled task is triggered internally and at that point needs access to the specific credentials stored before while reading these credentials is not necessary for an interactive user session, and should be prevented for anything but internal system operation.*

Since the stored credentials comprise sensitive information, unauthorized access should not only be prevented programmatically, but also cryptographically.

Requirement 3. *All content of the key management system must be stored in an encrypted form.*

5.6 IBM SmartCloud Archive — Security Evaluation

Based on the attack patterns identified in the beginning of this chapter, this section will evaluate the security of the IBM SmartCloud Archive by applying the attack patterns and identifying vulnerabilities.

User Interface

Cross-Site request Forgery (CSRF)

The evaluation has found that the portal application as well as eDiscovery manager and WorkplaceXT are susceptible to CSRF attacks. Any function these applications offer is triggered by calling a URL on the service layer with appropriate parameters. All these URL calls can be issued out of context, from any legitimate or malicious page running in the same browser. An attacker can exploit this vulnerability for example to create a new user with permissions of his choosing in the background just by having the victim visit a web page he prepared.

Cross-Site Scripting (XSS)

XSS vulnerabilities were found in the SCA portal application. As stated above, the XSS attack requires a call to the service layer that includes user data and returns the user data in the response in a format that allows the client web browser to execute JavaScript code.

Such a call was found in the `SaveDispositionSweepTask` method that is used to configure new disposition tasks. Besides the task parameters, it requires the user to provide a username and a password that is stored with the task definition. When saving the task, the provided user credentials are verified and an error message is displayed if they are invalid. This error message contains the user name (e.g. *The user name _ is invalid.*) and thus provides a mechanism for user supplied data to be returned, and injected into the context of the application. Enough special character are passed through and returned to the user interface unaltered to allow the execution of harmful JavaScript code. In the current implementation, the characters / and " are escaped and can't be used in the injected code. The " character can be replaced by other types of quote signs, while the / character is necessary to create HTML end-tags for JavaScript (`<script>...</script>`). This second obstacle is overcome by embedding JavaScript code within HTML tag-attributes that don't require a closing tag like the `<body>` element.

The `onload` attribute of the `<body>` element may contain JavaScript code that gets executed when the page is loading. It may contain an arbitrary number of JavaScript commands which can be used to read the SCA session-cookie and send it to the attacker.

Middleware

Since no SQL operations in SCA are generated containing user input, no SQL injection vulnerability could be identified.

The security of other middleware components is not directly infulenced by the SCA user interface since no direct access is possible aside from the external products Workplace XT and eDiscovery Manager.

Infrastructure Components

SCA is currently only distributed as a single-tenant offering on dedicated hardware, making a cross-tenant attack impossible. Due to the single-tenant design of the application, multi-tenancy is currently only possible on the middleware layer where the security relies entirely on the secure configuration of shared components, so no evaluation in this respect can be carried out on behalf of the SCA application itself.

The internal security currently relies on physically restricting access to the infrastructure components. All internal communication is secured through encrypted and authenticated communication channels, making a man-in-the-middle attack on network communication impossible.

Stored data is secured through operating system and middleware security features, preventing unauthorized access from an attacker with logical access to the infrastructure. Physical access to the stored data is not prevented since no encryption or secure key management is employed by the current implementation of SCA.

6 A Key Management System for IBM SmartCloud Archive

6.1 Architecture and Design

The proposed **Cloud Key Store (CKS)** solution relies on a client-server architecture to be secure and scalable in a large cloud environment. The server component manages the central key store and all its content while the client serves the application servers and contains no sensitive key material besides a client certificate used to authenticate against the server. This architecture allows hosting all sensitive material on a central server which can be more easily secured against physical access, and need not be available to services outside the cloud, providing physical separation of key material and encrypted content as well as a server component with a lightweight access protocol that can easily be secured.

Valid clients and their access permissions are only stored in a client certificate issued by a tenant administrator. This eliminates the need to directly interact with the server to set up clients and their permissions and allows shifting the responsibility of managing client access from the infrastructure provider to a service provider.

CKS is implemented as an Enterprise Java Bean using a Stateful Session configuration. Communication between the client and server component uses the Java RMI-IIOP protocol which allows for a small footprint of the client component as well as good interoperability with established middleware components and firewalls.

Terminology

Certain terms and abbreviations used in the context of this key management system will be introduced in the following.

Server: The server component of the key management system. It stores all key material and responds to requests from client instances.

Client: The client component of the key management system. It is used by an application to store and retrieve key material. The physical machine on which the client component runs is referred to as a server as well but is not to be confused with the *server component* of the key management system. In the context of this discussion, *server* always refers to the *server component*.

Tenant: Is used as a term for a customer of a cloud service. Data, application instances an key material is said to belong to a tenant, signifying that it belongs to a specific customer and must not be available to other customers.

Server Certificate: The public key certificate used by the server. It is also the root certificate of the Public Key Infrastructure used by the key management system.

Server Private Key (SPK): Refers to the private key belonging to the Server Certificate. Further Certificates and public key pairs are issued for each tenant and client. See Section Certificates for a detailed description.

Server Master Key (SMK): The SMK is used to de- and encrypt all other key material on the server. The relationship between the keys is described in the

Tenant Master Key (TMK): The TMK is used by the server to encrypt all key material belonging to a certain tenant. It is protected by the SMK.

Server Administrator: The responsibility of the Server Administrator are to set up new Tenants and manage the server configutation.

Tenant Administrator: The Tenant Administrator holds the Tenant Certificate and private key in order to introduce new clients, and revoke Client Certificates.

Client Component

The client component consists of a Java library which handles communication with the key management server and can be integrated into any Java application. The client library features a high-level API providing access to key material on the server while abstracting the authentication and communication process.

Server Component

The server component stores and manages the key server contents in an encrypted form, his server certificate private key as well as all the valid tenant certificates and a client revocation list. Three individual storage mechanisms are in use to meet these requirements. A database instance stores operational data including revoked clients, tenants and key material. The server certificate and the corresponding private key are stored in a PKCS12 key store which is secured with the server master key. General configuration data concerning database connectivity and certificate location is kept in a configuration file.

System Architecture

The components making up the CKS system and their relationship are shown in Figure 6.1. The Server component lives separated from the client in a JVM running on the server hardware in a secure location. The client application using the key server integrates a client library JAR that implements the communication protocol and provides high-level functions to access the key management server. For testing and development, a small servlet-based test client was implemented that can also be used to perform basic maintenance tasks like listing server content, and removing objects. Both server and client require access to private key of their respective certificate. On the server the private key is secured using the Server Master Key, on the client a passphrase is used. Access to the SMK and passphrase must be provided when starting the services. Methods for securely storing and providing these initial secrets are presented in Section 4.3.

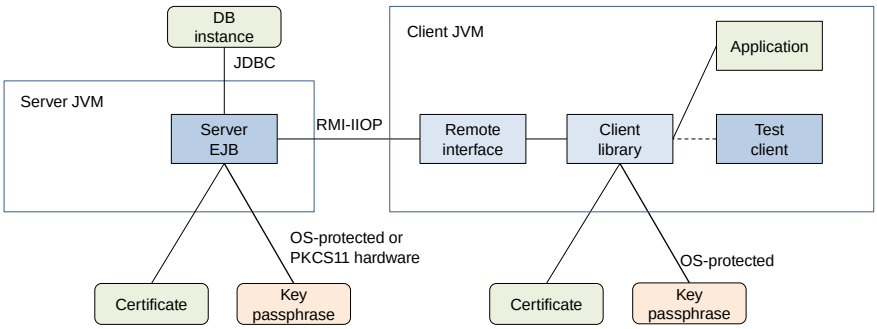

Figure 6.1: CKS — System Architecture

6.2 Use Cases

Before deciding on further design aspects of CKS like data structures and functions, the use cases for the key store have to be defined.

Following the requirements identified in chapter 4, and the design decisions stated in the beginning of this chapter, the following use cases should be supported by the key management system.

UC1 — Initializing the Server

The Server Administrator installs the server component and sets up the basic configuration and key material.

Success steps

- *Server Administrator:* Deploy the Server component on an application server.
- *Server Administrator:* Provide the configuration file and run the Server component's configuration tool to create the cryptographic material.
- *Server configuration tool:* Generate Server Master Key as well as certificate and protected private key.
- *Server Administrator:* store the SMK in the secure location discussed in Section 4.3.

UC2 — Setting up a Tenant Administrator

The Server Administrator configures a new tenant instace on the server and passes the cryptographic material on to the new Tenant Administrator.

Success steps

- *Server Administrator:* Use the Server configuration tool to create cryptographic material for the new Tenant and initialize the Tenant configuration.
- *Server configuration tool:* Generate a Tenant Certificate and private key and TMK. Output the Certificate and private key and store the configuration on the Server.
- *Server Administrator:* Securely pass the key material to the Tenant Administrator.

UC3 — Initializing a Client

A Tenant Administrator installs the client component and prepares it for operation by issuing a new Client Certificate.

Success steps

- *Tenant Administrator:* Issue a new Client Certificate containing the desired access permissions and sign it using the Tenant Private Key.
- *Tenant Administrator:* Deploy the client component on an application server with the new Client Certificate and private key.
- *Tenant Administrator:* Store a copy of the Client Certificate for use in client revocation.

UC4 — Revoking a Client

A Tenant Administrator removes a compromised Client by revoking the Client's certificate.

Success steps

- *Tenant Administrator:* Connect to the Server.
- *Server:* Authentication successful.
- *Tenant Administrator:* Transmit affected Client Certificate, request revocation.
- *Server:* Operation completed.

Error cases

- Authentication failure.

UC5 — Client to Server authentication

Before a Client instance can request operations on the Server, it needs to be authenticated through the 2-way authentication procedure described in Section 6.7.

As step one, this operation if followed by one of the use cases below, consituting a two-step self-contained operation.

Success steps

- *Client:* Request authentication, send own certificate.
- *Server:* Verify certificate signature with the stored Tenant certificate, enter Session Key negotiation phase.
- *Client:* Complete Session Key negotiation.
- *Server:* Authentication successful.

Error cases

- Authentication failure.
- Client Certificate is no longer valid, or revoked.

UC5.1 — Storing new key material on the Server

After successful authentication, a Client may store key material on the server depending on the predefined access permissions.

Success steps

- *Client:* Connect to the server.
- *Server:* Authentication successful.
- *Client:* Provide new object with alias and access policy, request insert operation.
- *Server:* Operation successful.

Error cases

- Client is not allowed to insert objects.
- Object alias is already taken.

UC5.2 — Requesting key material from the Server

After successful authentication, a client may request objects from the server depending the access policy stored in the Client Certificate.

Success steps

- *Client:* Connect to the server.
- *Server:* Authentication successful.
- *Client:* Request object by object alias.
- *Server:* Operation successful, object returned.

Error cases

- The Client is not allowed to access the object.
- The object does not exists.
- To prevent information disclosure, the Client should not be able to distinguish these error cases.

UC4.3 — Modifying key material on the Server

After successful authentication, a client may modify key material on the Server by providing the object alias, and the new value.

Success steps

- *Client:* Connect to the server.
- *Server:* Authentication successful.
- *Client:* Request modification by object alias and new value.
- *Server:* Operation successful.

Error cases

- The Client is not allowed to access the object.
- The object does not exists.
- To prevent information disclosure, the Client should not be able to distinguish these error cases.

UC5.5 — Remove an object from the Server

After successful authentication, a client may remove key material from the Server by providing the object alias and sufficient access permissions.

Success steps

- *Client:* Connect to the server.
- *Server:* Authentication successful.
- *Client:* Request removing an object by providing the object alias.
- *Server:* Operation successful.

Error cases

- The Client is not allowed to access the object.
- The object does not exists.
- To prevent information disclosure, the Client should not be able to distinguish these error cases.

6.3 Features

Following the use cases and the design guidelines stated above, the Cloud Key Store system will have the following features in order to fulfill the requirements.

Feature 1. *A physical separation between a client using, and a server storing and managing the key material will be provided.*

Feature 2. *Through an access scheme it is possible to assign clients the permission to create, read and write content.*

Feature 3. *The revocation of these permissions is supported.*

Feature 4. *Interaction with the server is minimized by delegating the task of introducing clients and defining their access permissions to a Tenant Administrator.*

Feature 5. *Clients need to authenticate against the server while the server is capable of identifying individual clients through the authentication and enforce access restrictions.*

Feature 6. *Due to the support of multi-tenancy, CKS allows for tenant separation through the use of separate logical name-spaces and separate cryptographic keys per tenant.*

Feature 7. *CKS uses a relational database for storing the key store content but employs an encryption scheme above the database layer. Therefore backup and high-availability features present in the DBMS can be harnessed by CKS without security concerns.*

6.4 Certificates

Public key certificates are used in CKS for three purposes. To verify the identity of communication partners, to negotiate symmetric encryption keys during communication and to specify access rights. This is achieved by issuing certificates to the server, each tenant and each client. The server is positioned as a certificate authority who possesses a root certificate which signs each tenant certificate. The tenant certificate is then used to sign client certificates, as seen in Figure 6.2.

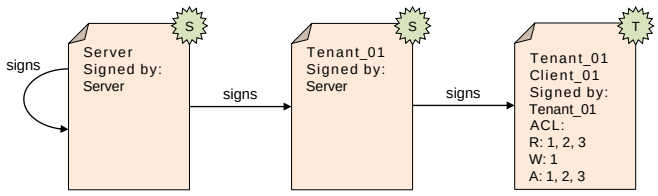

Figure 6.2: CKS — Certificate Relationship

Certificates can store identity, and additional information about the subject they are issued to. The server certificate does not necessarily need to contain any subject information while the tenant certificate contains a unique tenant ID that is used to pass on to client certificates, and later associate a client with a tenant. Besides this tenant ID, the client certificate contains a unique client ID for identification and auditing purposes. The certificate is also used to hold the access control list for the client, which cannot be manipulated since the whole certificate is signed by the tenant certificate. Through this mechanism, the server does not need to have a list of allowed clients and their access permissions, but can read this information from the client certificate during a key request. This allows for clients to be introduced to the server without direct interaction with the server, but by a tenant administrator alone who is in possession of the valid tenant certificate and the associated private key.

Revocation of client certificates is possible by the tenant administrator through a revocation list on the server which is checked by the server process on each client request.

This certification scheme requires the distribution of server, tenant and client certificates to the individual subject. The following shows the distribution of certificates, and their purpose.

- Server
  - *Server certificate & private key:* Verification of local tenant certificates, de- and encryption of stored keys, negotiation of communication session keys.
  - *Tenant certificates:* Verification of client certificates during key requests.
- Tenant administrator
  - *Tenant certificate & private key:* Signing of newly issued client certificates, identity verification against the server for client revocation requests.
  - *Server certificate:* Distribution to newly instantiated clients.
  - *Client certificates:* When revoking a client certificate, the administrator needs to know its hash value to communicate it to the server.
- Client instance
  - *Client certificate & private key:* Identity verification and session key negotiation during communication with the server.
  - *Server certificate:* Verification of the server identity, session key negotiation.

6.5 Encryption Scheme

CKS uses a combination of cryptographic and programmatic security in its operation to provide a strong security as well as reasonable administrative overhead. To provide tenant separation and security of data at rest all sensitive material is encrypted. For each tenant a symmetric encryption key (Tenant Master Key, TMK) is generated and used to encrypt all

key material belonging to this customer. All TMKs are themselves encrypted with the Server Master Key (SMK). The SMK is also used to secure the private key belonging to the server certificate. The SMK is kept in a secured memory location during operation and has to be provided on server startup either manually, read from storage, or from a secure hardware cryptographic module like a Trusted Platform Module (TPM)

a the private key of the server certificate. Access to this private key requires a secure pass phrase, the server master key (SMK) which is kept in memory during the operation of the CKS and has to be provided at start-up through one of the methods discussed in Section 4.3 depending on the security requirements.

6.6 Access Scheme

To allow a more flexible use of the key management system, different access patterns for each client instance are supported. Additionally the server should manage multiple tenants and allow a strict separation of access pattern between tenants.

This is achieved by placing the key material into *access groups*, and assigning clients to these groups with a set of permissions to read, write or store new objects. Access groups and objects (key material) are separated by tenant, and so are the clients. Each client certificate contains information about the access group membership which makes the definition of cross-tenant access paths impossible. A client accessing key material does not need to know to which group it belongs, the server checks if the client has the necessary access permission. When storing new key material, the client must specify to which access group it should belong.

Each client certificate contains three access control lists for reading, modifying and storing new objects. Each of these lists contain a set of access group identifiers.

The separation of duties between the Server and Tenant Administrator is achieved through this method since Tenant Administrators assign the ACLs to Client instances. A separation of duties on Clients is realized by the choice of access groups. Clients may have disjoint access group membership, preventing them from accessing the other Client's key material. Relationship types (e.g. Producer - Consumer) can be modelled with access groups and ACLs by assigning different permissions for the same group.

Figure 6.3 shows a model of this access scheme.

6.7 Communication Protocol

The communication between client and server uses the RMI-IIOP protocol and employs a custom, integrated authentication and encryption mechanism. This secures the data being transmitted beyond any lower level security mechanism like SSL and reduces the need to implement such an infrastructure, avoiding an insecure configuration.

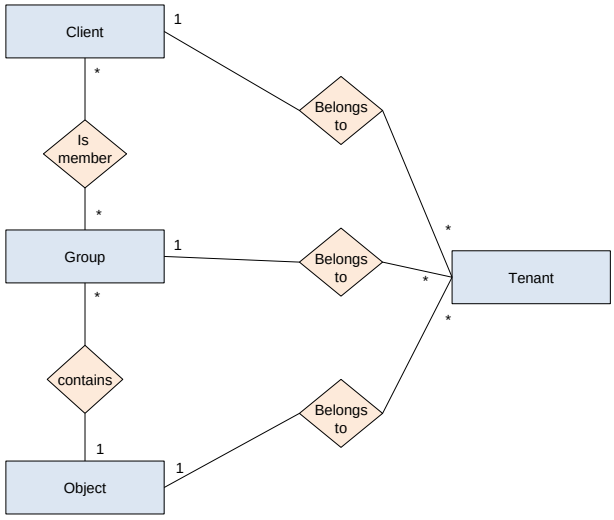

Figure 6.3: CKS — Access Scheme

All key requests constitute a complete, self-contained operation and require authentication before the request is answered. The authentication mechanism is based on two-way authentication and a fast challenge-response scheme to negotiate a symmetric session key used for encrypting the content of the communication. The client starts by transmitting his certificate to show his identity to the server, and a random number used in the creation of the session key. The server verifies the identity of the certificate by checking the signature, and responds with the random number generated by the client and a new one generated by the server. The client verifies his random number and thereby verifies the server's identity. He responds with the server's random number, verifying his identity towards the server. The now exchanged random number allow for the computation of a session key on both sides.

See Figure 6.4 for the sequence of events in this authentication process.

6.8 Data Structure

The server component of the key management system needs to store configuration data to recognize clients, authorize requests, and manage the key store. This data is stored in a relational database on the server which allows delegation of functionality to the database and the aforementioned easy backup/recovery as well as high availability through replicated database instances.

The database schema needed to support the required functionality is shown below. One database is used for storing information about tenants and their clients to tie a request to the specific tenant before checking permissions.

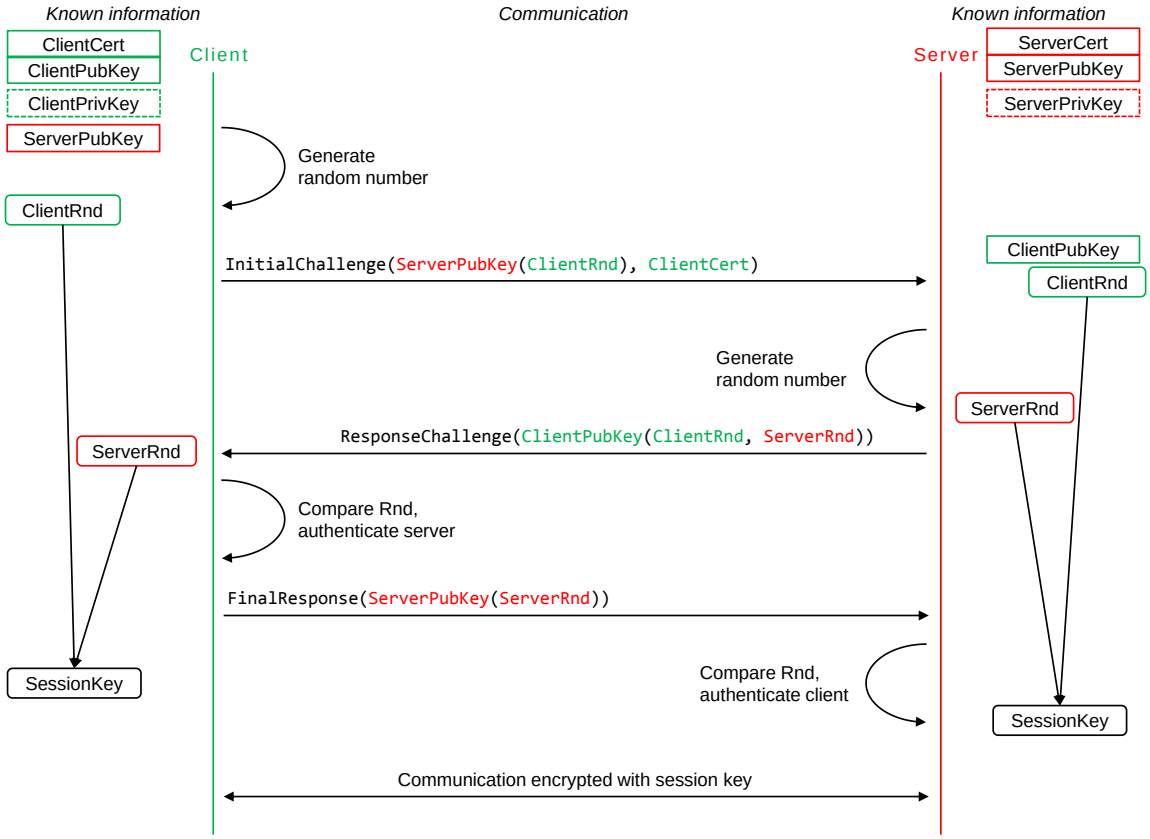

Figure 6.4: CKS — Two-Way Authentication

Another database contains hash values of revoked client certificates. A dedicated database is assigned to each tenant, storing the specific access rules and the actual key material.

```
Database ServerConfiguration:
Tenants:             (TenantID, Name)
TenantCertificates: (TenantID, TenantCertificate)
TenantMasterKeys:    (TenantID, TMK)
RevokedClients:      (TenantID, ClientCertificateHash)


Database TenantID:
Groups:              (GroupID, Description)
Objects:             (ObjectAlias, ObjectReference)
GroupObjects:        (GroupID, ObjectAlias)
```

To assure data integrity of the server configuration, these constraints were defined to enforce the database schema.

- When deleting a row in `Tenants`, delete all rows in `TenantCertificates`, `TenantMasterKeys`, `RevokedClients` belonging to that tenant and delete the `TenantID` database.
- Only allow deletion of `groups` if no more *objects* are referencing it.
- Deletion of *GroupObjects* is not allowed.
- When deleting a row in `Objects`, delete all rows in `GroupObjects` referencing that object.

7 Conclusion and Outlook

This thesis has shown the vast amount of different service configurations possible in a cloud environment, and has analyzed the security implications of various design aspects. It was pointed out that the combinations of cloud deployment models, service models and tenant separation models create a diverse landscape of possible cloud services with individual security requirements and possibilities for attack vectors.

While cloud services are comprised of existing technologies and solutions, it was shown that they not only inherit security vulnerabilities from these components, but create new surface area for novel attacks geared specifically towards cloud computing.

The concept of the cross-tenant attack takes existing attack vectors and applies them to this new environment, creating threats of higher severity than before. This multi-tenant property of cloud computing is the driving force behind all new threats towards cloud computing, and relies on the basic premise of the cloud to share components and infrastructure amongst customers in order to create affordable, high-quality services.

Due to these implications, the concept of multi-tenancy remains the most controversial in cloud computing and the number one factor for customers to go in the direction of private clouds and single-tenant configurations.

This ongoing trend contradicts the premise of cloud computing in that it relinquishes the advantages that can be achieved by harnessing the economy of scale seen in large-scale consumer cloud applications like social networks or email services.

In order for enterprise customers to accept the multi-tenant concept and take full advantage of cloud computing, sophisticated security concepts that improve tenant isolation are necessary.

The distinction made in this thesis between logical and physical access to the system under the light of a shared infrastructure shows that the severity attributed to a malicious insider rises significantly when the physical infrastructure is no longer under the control of the customer.

While the system can be defended against logical access and remote attackers with conventional measures, the protection against an attacker with physical access under the light of a multi-tenant infrastructure raises a new field of study that can only party benefit from established security measures.

In order to provide a basic set of tools for creating such a multi-tenant environment secured against malicious insiders, the concept of encryption was introduced. The basic mechanism

of encryption to reduce the amount of sensitive information from gigabytes down to a small encryption key was used throughout the proposed security concept.

The basic premise for this rationale is the fact that ultimately data security relies on physically securing some system resources against unauthorized access. To make this approach feasible, the sensitive-information reducing property of encryption was used to first reduce large amounts of stored sensitive data to a number of encryption keys, which are further reduced to an authentication key.

This reduction process created dependencies between the involved objects data, encryption key and authentication key. These objects can now be physically separated while their dependency remains preserved. The physical separation of these objects allows fulfilling the requirement of physically securing sensitive information.

To achieve this goal, the concept of a central key management system was introduced to play the part of the only physically secured infrastructure component in the system. It was shown that this approach lacks widespread support in applications and services, which lead to the implementation of the prototypical Cloud Key Store system.

Under the light of physical security, it was analyzed how interaction with this crucial component can be minimized in order to reduce the surface area for possible attacks.

The concept of certificate based authorization was introduced as a means of further delegating responsibilities away from the central key management component, to trusted tenant administrators.

It was shown that the proposed security design involving a central key management system presents a possibility to secure a multi-tenant cloud environment against malicious insiders under less strict conditions than previous approaches. This suggests further research in the central component of this approach, the key management server and client component.

While vendor-specific proprietary solutions for such a scenario exist in some commercial encryption products, further research in this topic is necessary in order for this approach to be widely recognized as a trustable solution for enterprise cloud environments.

An important factor for wide-spread adoption of such a system is not only its reputation, but also its availability. It is necessary to have a common standard and interoperability protocol for client-server communication with the central key management component.

Once these conditions are met and widespread support for encryption and key management solutions for a cloud environment exist, it will be possible for cloud providers to create a service with unparalleled data security only comparable to conventional on-premise IT-systems, eliminating the last reservations customers have towards the adoption of cloud computing.

Bibliography

[AAB+05] W. J. Armstrong, R. L. Arndt, D. C. Boutcher, R. G. Kovacs, D. Larson, K. A. Lucke, N. Nayar, R. W. Swanberg. Advanced virtualization capabilities of POWER5 systems. *IBM Journal of Research and Development*, 49(4.5):523 –532, 2005. doi:10.1147/rd.494.0523. (Cited on page 11)

[AB11] C. R. Andrey Bogdanov, Dmitry Khovratovich. Biclique Cryptanalysis of the Full AES. Technical report, Microsoft Research Redmond, USA, 2011. (Cited on page 28)

[Amo94] E. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994. (Cited on pages 16, 37 and 69)

[And01] R. Anderson. *Security Engineering*. John Wiley & Sons, Chichester, 2001. (Cited on pages 27 and 28)

[ASR09] M. Alomari, K. Samsudin, A. Ramli. A Study on Encryption Algorithms and Modes for Disk Encryption. In *2009 International Conference on Signal Processing Systems*, pp. 793 –797. 2009. doi:10.1109/ICSPS.2009.118. (Cited on page 25)

[BEE+10] J. Bacon, D. Evans, D. M. Eyers, M. Migliavacca, P. Pietzuch, B. Shand. Enforcing end-to-end application security in the cloud (big ideas paper). In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, pp. 293–312. Springer-Verlag, Berlin, Heidelberg, 2010. URL http://dl.acm.org/citation.cfm?id=2023718.2023739. (Cited on page 35)

[BS99] J. K. Bruce Schneier. Secure Audit Logs to Support Computer Forensics. *Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419*, 1999. URL https://www.schneier.com/paper-auditlogs.pdf. (Cited on page 22)

[bsi10] Security Recommendations for Cloud Computing Providers. Technical report, Federal Office for Information Security (BSI), 2010. URL https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Minimum_information/SecurityRecommendationsCloudComputingProviders.pdf. (Cited on page 49)

[Bö11] A. Börner. *Orchestration and Provisioning of Dynamic System Topologies*. Master's thesis, University of Stuttgart, 2011. (Cited on page 19)

[CGJ+09] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*,

CCSW '09, pp. 85–90. ACM, New York, NY, USA, 2009. doi:http://doi.acm.org/10.1145/1655008.1655020. URL http://doi.acm.org/10.1145/1655008.1655020. (Cited on page 10)

[CSA10] CSA. Top Threads to Cloud Computing. Technical report, Cloud Security Alliance, 2010. URL http://www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf. (Cited on pages 49 and 65)

[CSS+09] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra, D. Zamboni. Cloud security is not (just) virtualization security: a short paper. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pp. 97–102. ACM, New York, NY, USA, 2009. doi:http://doi.acm.org/10.1145/1655008.1655022. URL http://doi.acm.org/10.1145/1655008.1655022. (Cited on page 65)

[DH76] W. Diffie, M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644 – 654, 1976. doi:10.1109/TIT.1976.1055638. (Cited on page 17)

[DLFMT04] G. Di Lucca, A. Fasolino, M. Mastoianni, P. Tramontana. Identifying cross site scripting vulnerabilities in Web applications. In *Web Site Evolution, 2004. WSE 2004. Proceedings. Sixth IEEE International Workshop on*, pp. 71 – 80. 2004. doi:10.1109/WSE.2004.10013. (Cited on page 53)

[DMLW08] S. M. Diesburg, C. R. Meyers, D. M. Lary, A.-I. A. Wang. When cryptography meets storage. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, StorageSS '08, pp. 11–20. ACM, New York, NY, USA, 2008. doi:http://doi.acm.org/10.1145/1456469.1456472. URL http://doi.acm.org/10.1145/1456469.1456472. (Cited on page 25)

[DMT11] K. Dahbur, B. Mohammad, A. B. Tarakji. A survey of risks, threats and vulnerabilities in cloud computing. In *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications*, ISWSA '11, pp. 12:1–12:6. ACM, New York, NY, USA, 2011. doi:http://doi.acm.org/10.1145/1980822.1980834. URL http://doi.acm.org/10.1145/1980822.1980834. (Cited on page 51)

[eni10] Cloud Computing Risk Assessment, 2010. URL https://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment. (Cited on pages 49 and 65)

[Fri11] F. Fritz. *Maximization of resource utilization through dynamic provisioning and deprovisioning in the cloud*. Master's thesis, University of Stuttgart, 2011. P. 11. (Cited on pages 14 and 19)

[Gol97] S. Goldwasser. New directions in cryptography: twenty some years later (or cryptograpy and complexity theory: a match made in heaven). In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pp. 314 –324. 1997. doi:10.1109/SFCS.1997.646120. (Cited on page 17)

[HAF+07] G. Hunt, M. Aiken, M. Fähndrich, C. Hawblitzel, O. Hodson, J. Larus, S. Levi, B. Steensgaard, D. Tarditi, T. Wobber. Sealing OS processes to improve dependability and safety. *SIGOPS Oper. Syst. Rev.*, 41:341–354, 2007. doi:http://doi.acm.org/10.1145/1272998.1273032. URL http://doi.acm.org/10.1145/1272998.1273032. (Cited on page 63)

[Har07] L. Hars. Discryption: Internal Hard-Disk Encryption for Secure Storage. *Computer*, 40(6):103 –105, 2007. doi:10.1109/MC.2007.202. (Cited on page 24)

[IT10] D. W. S. Iryna Tsvihun, Philipp Stephanow. Vergleich der Sicherheit traditioneller IT-Systeme und Public Cloud Computing Systeme. Technical report, Fraunhofer Institut, 2010. (Cited on page 11)

[KAAS11] K. Khan, M. Amin, A. Afridi, W. Shehzad. SELinux in and out. In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pp. 339 –343. 2011. doi:10.1109/ICCSN.2011.6014064. (Cited on page 46)

[KN08] S. H. P. Karsten Nohl, David Evans. Reverse-Engineering a Cryptographic RFID Tag. *USENIX Security Symposium*, 2008. (Cited on page 18)

[LC10] M. Liang, C. wen Chang. Research and design of full disk encryption based on virtual machine. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 2, pp. 642 –646. 2010. doi:10.1109/ICCSIT.2010.5565144. (Cited on page 25)

[NJ05] A. Nadeem, M. Javed. A Performance Comparison of Data Encryption Algorithms. In *Information and Communication Technologies, 2005. ICICT 2005. First International Conference on*, pp. 84 – 89. 2005. doi:10.1109/ICICT.2005.1598556. (Cited on page 28)

[Pav11] D. Pavlovic. Gaming security by obscurity. In *Proceedings of the 2011 workshop on New security paradigms workshop*, NSPW '11, pp. 125–140. ACM, New York, NY, USA, 2011. doi:http://doi.acm.org/10.1145/2073276.2073289. URL http://doi.acm.org/10.1145/2073276.2073289. (Cited on page 18)

[Pha10] H. Pham. User interface models for the cloud. In *Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pp. 359–362. ACM, New York, NY, USA, 2010. doi:http://doi.acm.org/10.1145/1866218.1866223. URL http://doi.acm.org/10.1145/1866218.1866223. (Cited on page 11)

[PM11] T. G. Peter Mell. The NIST Definition of Cloud Computing, 2011. (Cited on page 12)

[RSA83] R. L. Rivest, A. Shamir, L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26:96–99, 1983. doi:http://doi.acm.org/10.1145/357980.358017. URL http://doi.acm.org/10.1145/357980.358017. (Cited on page 18)

[Sav11] L. Savu. Cloud Computing: Deployment Models, Delivery Models, Risks and Research Challenges. In *Computer and Management (CAMAN), 2011 International Conference on*, pp. 1 –4. 2011. doi:10.1109/CAMAN.2011.5778816. (Cited on page 50)

[Sha49] C. Shannon. Communication Theory of Secrecy Systems. Technical report, Bell System Technical Journal 28(4):656–715, 1949. (Cited on page 17)

[SKS11] S. Sengupta, V. Kaulgud, V. Sharma. Cloud Computing Security–Trends and Research Directions. In *Services (SERVICES), 2011 IEEE World Congress on*, pp. 524 –531. 2011. doi:10.1109/SERVICES.2011.20. (Cited on page 7)

[SS94] R. Sandhu, P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40 –48, 1994. doi:10.1109/35.312842. (Cited on page 34)

[sta09a] Information Technology Security Standards, 2009. URL https://w3-03.ibm.com/transform/sas/as-web.nsf/ContentDocsByTitle/ITCS104. (Cited on pages 50 and 57)

[sta09b] PKCS 11 Base Functionality v2.30: Cryptoki, 2009. (Cited on page 47)

[SV11] M. Siddiqui, D. Verma. Cross site request forgery: A common web application weakness. In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pp. 538 –543. 2011. doi:10.1109/ICCSN.2011.6014783. (Cited on page 56)

[Tom07] S. Tomasco. IBM Cloud computing, 2007. URL https://www-03.ibm.com/press/us/en/presskit/29681.wss. (Cited on page 19)

[XF10] I. X-Force. IBM X-Force® 2010 Trend and Risk Report. Technical report, IBM Security Solutions, 2010. URL https://www14.software.ibm.com/webapp/iwm/web/signup.do?source=swg-spsm-tiv-sec-wp&S_PKG=IBM-X-Force-2010-Trend-Risk-Report. (Cited on pages 52 and 63)

[XF11] I. X-Force. IBM X-Force® 2011 Mid-year Trend and Risk Report. Technical report, IBM Security Solutions, 2011. URL https://public.dhe.ibm.com/common/ssi/ecm/en/wgl03009usen/WGL03009USEN.PDF. (Cited on pages 52 and 57)

All links were last followed on March 06, 2012.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Tim Waizenegger)