Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

A Security Concept for Distributed Data Processing Systems

Julian Ziegler

Course of Study:	Informatik
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Dipl.-Inf. Pascal Hirmer
Commenced:	February 7, 2017
Completed:	August 7, 2017
CR-Classification:	H.3.4, H.4.0, K.6.5

Abstract

Today, the amount of raw data available is abundant. As only a small part of this data is in a form fit for further processing, there is many data left to analyze and process. At the same time, cloud services are ubiquitous and allow even small businesses to perform large tasks of distributed data processing without the significant costs required for a suitable computational infrastructure. However, as more and more users transfer their data into the cloud for processing and storage, concerns about data security arise. An extensive review of data security research in today's cloud solutions confirms these concerns to be justified. The existing strategies for securing one's data are not adequate for many use cases. Therefore, this work proposes a holistic security concept for distributed data processing in the cloud. For the purpose of providing security in heterogeneous cloud environments, it statically analyzes a data flow prior to execution and determines the optimal security measurements. Without imposing strict requirements on the cloud services involved, it can be deployed in a broad range of scenarios. The concept's generic design can be adopted by existing data processing tools. An exemplary implementation is provided for the mashup tool FlexMash. Requirements, such as data confidentiality, integrity, access control, and scalability were evaluated to be met.

Kurzfassung

Die heutige Menge an vorhandenen Daten ist enorm. Viele davon müssen zunächst verarbeitet und analysiert werden, da nur ein geringer Teil dieser Daten für die weitere Verarbeitung geeignet ist. Cloud-basierte Dienste sind allgegenwärtig und erlauben es auch kleineren Unternehmen Datenverarbeitung durchzuführen, ohne die Kosten von notwendiger Infrastruktur tragen zu müssen. Mit einer zunehmenden Zahl an Nutzern von Clouds wachsen jedoch auch Bedenken der Sicherheit. Eine ausführliche Durchsicht der aktuellen Forschung zu diesem Thema bestätigt diese Bedenken und existierende Strategien zur Sicherung der eigenen Daten berücksichtigen viele Fälle nicht. Daher stellt diese Arbeit ein ganzheitliches Sicherheitskonzept für die verteilte Datenverarbeitung in der Cloud vor. Damit Sicherheit in heterogenen Cloudumgebungen gewährleistet werden kann, wird ein Datenfluss vor der Ausführung statisch analysiert und es werden die für diesen Fluss optimalen Sicherheitsmaßnahmen festgelegt. Das Konzept besitzt einen breiten Anwendungsbereich, da keine straffen Anforderungen an die genutzten Dienste gestellt werden. Das generische Design des Konzepts ermöglicht eine einfache Integration in bereits existierende Datenverarbeitungsanwendungen, wie beispielhaft an FlexMash gezeigt wird. Anforderungen, wie die Vertraulichkeit von Daten, deren Integrität, Zugriffskontrolle und Skalierbarkeit des Systems konnten erreicht werden.

Contents

List of Figures

List of Abbreviations

AACS Advanced Access Content System. 30

BPMN Business Process Model and Notation. 16

CA Certificate Authority. 37

CCA Chosen-Ciphertext Attack. 29

CP-ABE Ciphertext-Policy Attribute-Based Encryption. 27

CSS Content Scramble System. 30

DFA Deterministic Finite Automata. 29

DNS Domain Name System. 21

DoS Denial-of-Service. 21

DPD Data Processing Description. 15

DSD Data Source Description. 15

EC2 Elastic Compute Cloud. 25

FDE Full Disk Encryption. 36

FTP File Transfer Protocol. 21

HDFS Hadoop Distributed File System. 36

HLA Homomorphic Linear Authenticator. 31

HTTP Hypertext Transfer Protocol. 21

IaaS Infrastructure as a Service. 22

IDEA International Data Encryption Algorithm. 36

IFC Information Flow Control. 35

KMIP Key Management Interoperability Protocol. 29

KVM Kernel-based Virtual Machine. 33

MAC Media Access Control. 24

MAC Message Authentication Code. 31

MITM Man-In-The-Middle. 20

MPC (Secure) Multi-Party Computation. 28

NaCL Google Native Client. 33

OASIS Organization for the Advancement of Structured Information Standards. 29

PaaS Platform as a Service. 22

PAP Policy Administration Point. 53

PDP Policy Decision Point. 53

PEP Policy Enforcement Point. 53

PIP Policy Information Point. 53

PKI Public Key Infrastructure. 26

PRE Proxy Re-Encryption. 29

PRP Policy Retrieval Point. 53

RPC Remote Procedure Call. 21

SaaS Software as a Service. 22

SAML Security Assertion Markup Language. 50

SOA Service-Oriented Architecture. 31

SOAP Simple Object Access Protocol. 21

SSH Secure Shell. 37

SSL/TLS Secure Sockets Layer/Transport Layer Security. 37

SSO Single Sign-On. 37

TCCP Trusted Cloud Computing Platform. 32

TOSCA Topology and Orchestration Specification for Cloud Applications. 52

TVMM Trusted Virtual Machine Monitor. 32

VM Virtual Machine. 23

VMM Virtual Machine Monitor/Hypervisor. 23

WS-BPEL	Web Services Business Process Execution Language. 16
XACML	eXtensible Access Control Markup Language. 37
XML	Extensible Markup Language. 21

1 Introduction

The days of constrained data storage are long over and, as nowadays acquiring data is no problem anymore either, the amount of data stored is constantly increasing [KAF+08]. However, Gantz and Reinsel [GR12] assume that less than 3% of data is analyzed and tagged. The remainder of this data is raw data, that is, it needs to be processed before it can be put to use. Out of this raw data, they estimated 23% to be useful to enterprises and other users. Consequently, there is a lot of data left to process.

Accessing this immense amount of raw data requires efficient analysis. Automatic processing, without human interaction, comes to mind and can be a suitable approach to make this data useful. However, as Puolamäki et al. [PBT+10] mention, humans have to be part of data analysis, in order for the analysis to cope with the complexity involved in processing domain-specific and heterogeneous data sets. Because data analysis and processing requires both technical experts and experts with domain-specific knowledge for a given set of data, this generates new problems, such as a high communication overhead and inflexible implementations. Solutions, such as FlexMash [HM16], are available to solve this and allow domain-specific users to create data mashups in order to realize data processing without technical knowledge.

Even though huge amounts of data are available and means to process them exist, the high costs for necessary computing infrastructures excluded many users in the past. However, nowadays cloud services for data processing and storage are ubiquitous, many offerings exist and considerably increase the audience of distributed data processing. As more and more users entrust their data to the cloud, the concern for security rises.

To address these concerns, a security concept for distributed data processing in the cloud is proposed. Its requirements are based on an extensive literature research to not only assess the security issues found in current cloud solutions but also to evaluate existing strategies to secure a user's data. The proposed solution of this work is a holistic security concept that allows a customer of cloud solutions to perform secure distributed data processing in an inhomogeneous environment of cloud services. Its design is generic and can be adapted to many existing data processing tools, as it is demonstrated with FlexMash.

Outline

This thesis is structured as follows:

Chapter 2 – Related Work and Fundamentals introduces the data mashup tool FlexMash and explains two selected cryptosystems.

Chapter 3 – Cloud Security: State of the Art provides an overview of security issues one has to expect as customer of cloud solutions, along with solutions to these issues found in current research.

Chapter 4 – Security Concept presents the proposed solution of this thesis to secure distributed data processing in the cloud.

Chapter 5 – Implementation in FlexMash suggests a way to implement the security concept in an existing data processing tool.

Chapter 6 – Evaluation validates whether the requirements of the security concept could be met.

Chapter 7 – Conclusion and Future Work finishes this thesis by summarizing the results and giving pointers for future work.

2 Related Work and Fundamentals

This chapter introduces FlexMash, which is used to demonstrate the proposed security concept, followed by the fundamentals of two different kinds of cryptosystems.

2.1 FlexMash

FlexMash is a data mashup tool that allows domain experts, without technical understanding of databases, clouds and services, to create complex data processing scenarios. It was first presented by Hirmer and Mitschang [HM16] and further extended by Hirmer and Behringer [HB17].

FlexMash consists of six steps. During the first step, a user models a *mashup plan*. It consists of so called Data Source Description (DSD) and Data Processing Description (DPD) nodes that a user can connect arbitrarily in order to create the desired data mashup. A user can create such a mashup using a graphical modeling tool without expertise in data processing. The modeler does not need any knowledge of how these

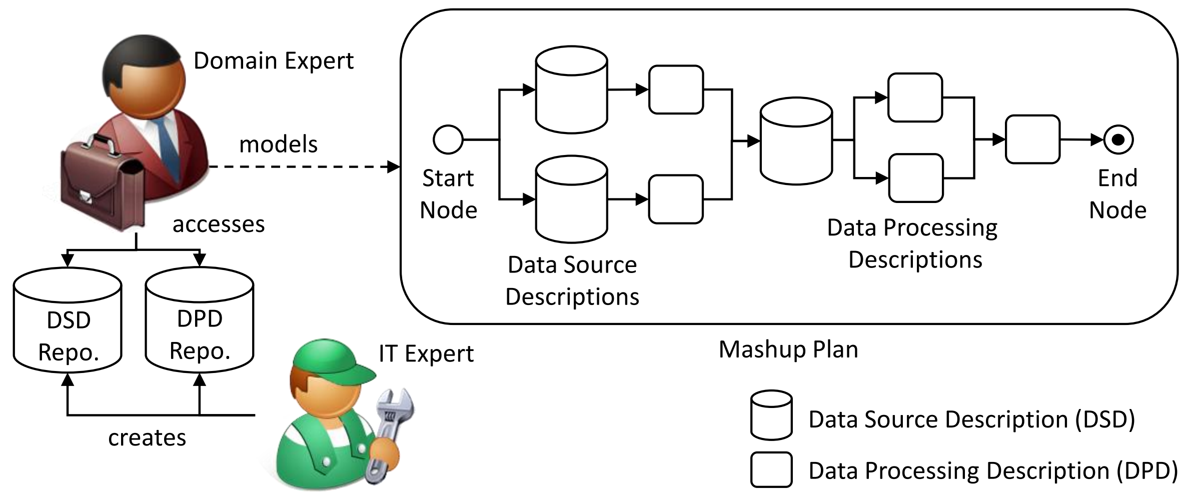


Figure 2.1: FlexMash overview with a domain expert modeling a mashup plan. Illustration by Hirmer and Mitschang [HM16].

nodes are implemented because implementations are provided by technical experts and stored in repositories, as can be seen in Figure 2.1 [HM16]. The result is an abstract description of a mashup — a plan that cannot be executed yet.

In the cours of the next step, the modeler can select a set of non-functional requirements from a catalog. These requirements can influence data processing in different ways, such as prioritizing robustness or execution speed.

During the third step, a runtime environment has to be selected, which is mainly influenced by the previous selection of requirements. Communication layers, interfaces between software components, and the execution engine, are part of the runtime environment and need to be chosen so that the selected requirements can be fulfilled.

The fourth step transforms the abstract and general plan into an executable form that can be directly used by an appropriate engine. Possible targets for an executable plan include the Business Process Model and Notation (BPMN) standard and the Web Services Business Process Execution Language (WS-BPEL), both representations for business processes. These languages enjoy widespread software support and allow the usage of readily available workflow technology for execution.

The chosen engine executes the transformed plan during the fifth step by provisioning appropriate services on-demand for each DSD and DPD node. Their respective implementation extracts data from defined sources and processes it according to the modeled plan.

The final step completes the data processing by either visualizing results or using them as input for further processing. These steps can also be seen in Figure 5.1 [HB17].

2.2 Cryptosystems

Cryptography is an important technology in the field of information security, with application to secure cloud data processing, as explained by Winkler [Win11]. It can be used to prevent adversaries from reading confidential messages and, to an extent, impede modifications. Cryptosystems can be divided into two major categories: symmetric and asymmetric ones. In a symmetric cryptosystem, it is computationally easy to derive a decryption key from the corresponding encryption key (and vice versa). In many cases they are even identical. However, in an asymmetric scheme, it is computationally not feasible to derive a private key based on a public key.

Further variations exist in these categories of cryptosystems. Following, two asymmetric cryptosystems with useful features are explained in greater detail.

2.2.1 Proxy Re-Encryption

The concept of proxy re-encryption was introduced by Blaze, Bleumer, and Strauss [BBS98]. It describes a public-key cryptosystem that allows a third party (proxy) to alter a chipertext that has been encrypted for a user's key pair, so that another user's key pair can decrypt it. The goal is to achieve this without having plaintext accessible at any time of the re-encryption and without revealing the secret key of involved users — security is not to be compromised.

In an asymmetric encryption scheme used by the users A and B, each user has a secret key SK and a public key PK. The functions $E()$ and $D()$ perform encryption and decryption respectively, so that $m = D_{SK_A}(E_{PK_A}(m))$ holds true. Proxy re-encryption adds a re-encryption key $RK_{A \to B}$ and a re-encryption function $\pi()$ that allows us to transform the encryption of a given ciphertext from one user (delegator) to another (delegatee). These additions allow:

$$m = D_{SK_B}(\pi_{RK_{A \to B}}(E_{PK_A}(m)))$$

The re-encryption can be delegated to a third party because the function $\pi()$ requires no cryptographic key from A or B, only the specifically generated re-encryption key $RK_{A \to B}$.

For ideal security, a theoretical proxy re-encryption scheme should prevent a proxy from reading any plaintext. Neither delegator, delegatee nor proxy should be able to see secret keys of each other. The generation of a re-encryption key should require an active delegate and the key should work unidirectional, that is, a generated key $RK_{A \to B}$ cannot be used in reverse $RK_{A \leftarrow B}$. Moreover, B should be unable to further delegate A's re-encrypted messages.

Chung, Liu, and Hwang [CLH14] list various properties that can be used to classify proxy re-encryption schemes, for example, the level of trust placed into the proxy. In addition, the already mentioned bi- or unidirectionality of a delegation and whether generating the re-encryption key requires an active delegate. The delegation can be transitive and can allow re-delegation by the delegatee. Furthermore, it can be temporary, so that re-encryption is only possible until a certain event occurs.

The proxy can decide for whom it performs re-encryption of certain data. Additionally, A can decide if a re-encryption key $RK_{A \to B}$ can be generated and the proxy can decide whether B can access it.

2.2.2 Homomorphic Encryption

In a homomorphic encryption scheme calculations can be performed directly on ciphertext with the decrypted result equaling a calculation performed on the plaintext. With $E_k()$ being an encryption function using key k, we have:

$$E_k(f_1(m_1, m_2)) = f_2(E_k(m_1), E_k(m_2))$$

Note that neither of the functions f_1, f_2 require any additional input such as cryptographic keys. This property is especially interesting whenever a third party wants to work on encrypted data without the need for decryption, key transfers, and possible violation of security.

The general term of homomorphic cryptosystems can be subdivided into fully and partially homomorphic cryptosystems. A fully homomorphic cryptosystem allows to perform arbitrary computations on ciphertext any number of times, while partially homomorphic systems can only perform certain operations, such as addition or multiplication. Fully homomorphic systems come with a huge computational cost and real time usage in larger scales is questionable at this point [NLV11]. Effort is made to change this and more efficient schemes do exist [BGV12] as well as usable implementations [Hal17].

Many more widely known cryptosystems have partially homomorphic properties, for example, RSA and ElGamal [El 85] [1], both homomorphic with respect to multiplication.

[1]An asymmetric cryptosystem. Used, among others, by *GNU Privacy Guard*

3 Cloud Security: State of the Art

As cloud offerings are applied to more and more scenarios, the concern for security increased equally among already acquired as well as potential customers. Naturally, security related challenges are object of ongoing research. Following, security concerns of cloud offerings are presented and existing solutions are evaluated.

3.1 Security Concerns

From low level virtualization to high level protocols, the cloud utilizes a broad range of technologies to provide attractive services. While customers benefit from the advantages, they are also exposed to the combined amount of security issues of the involved technologies [GWS11]. The increased amount of network traffic, compared to local servers, adds further attack vectors a customer of cloud offerings is exposed to.

Before general security concerns can be discussed, a distinction has to be made between the concept of *threats* and *vulnerabilities*, based on a definition by Hashizume et al. [HRFF13]. While threats represent a specific attack — a malicious action taken against a victim — a vulnerability is the means through which such an attack can be carried out. A threat exploits a vulnerability. Each vulnerability comes along with potential threats and a single threat can be realized through different vulnerabilities or a combination thereof. Even though it is possible to detect the occurrence of a threat by means of audit and monitoring, a threat itself cannot be fixed. Instead, the corresponding vulnerability has to be eliminated. As an example, one might consider a personal computer without password protection. Everyone with physical access can extract personal data from such a computer and this action represents a threat. In order to prevent such a threat, the underlying vulnerability — lack of password protection — has to be eliminated. However, even though implementing a password will eliminate this vulnerability, there might still be others present that allow an attacker to extract private data. As such, threats are useful to identify vulnerabilities and help to asses the security of a system. However, removing one vulnerability does not necessarily eliminate a threat.

In the following, general security concerns in distributed data processing are presented in form of real and existing threats, along with the vulnerabilities they lead to. Concerns are categorized in order to provide a better overview. Instead of a classification strictly enumerating technical components, categories are based on the coverage of existing research to allow for a problem-centric approach.

Data Security

Threats concerning data are potentially most important to users and are present in every area of data processing. They endanger confidentiality and integrity of a cloud offering customer's data and affect data in all situations: when being transmitted (*in-transit*), saved on non-volatile storage (*at-rest*) and even while being processed (*in-use*).

Essentially, all unencrypted data has to be considered vulnerable to threats targeting confidentiality. Data without digital signatures or inadequate communication security is susceptible to forged authenticity and loss of integrity. Such can be accomplished with Man-In-The-Middle (MITM) attacks [Wil14], allowing an adversary to eavesdrop or even manipulate data in-transit.

Due to the managed nature of cloud solutions, various tasks, such as backups, are often transparently done by the cloud provider, sometimes even delegated to contracted third parties. As Jansen [Jan11] explains, this can introduce further vulnerabilities a user might not even be aware of. This also turns secure deletion of data into a difficult problem because a user does not necessarily know whether more copies of his or her data exist, nor where they are.

Access Management

When handling data, it is important to know who is allowed to access and alter which parts of a data set. In order to enforce access policies, it is necessary to properly identify and authenticate users as well as computer systems. The absence of access management, identification and authorization as well as key management systems, are vulnerabilities itself. However, not only do such systems have to exist in the first place, they also have to be secure and need to be used correctly to prevent introduction of new, system-specific vulnerabilities.

Especially with cloud offerings that usually provide at least one privileged account to customers, Hashizume et al. [HRFF13] point out account or service hijacking as a serious concern because it gives an attacker complete control. Grobauer, Walloschek, and Stocker [GWS11] bring up the similarly threatening session riding: typically exploiting weaknesses in a cloud provider's management of user sessions, granting an attacker privileges equal to the victim's account. Furthermore, impersonation of privileged users is a threat in the absence of identification and authentication mechanisms.

A key management system is often a crucial part of access management. In order to implement such systems, multiple questions have to be answered, such as: way of key exchange, location of a key store and mapping of access privileges to it. This is no easy task, since an architectural flaw can expose vulnerabilities that render such a system ineffective or even useless. This is a problem, especially in cloud environments, where few approaches or open standards exist for key management systems, as Chandramouli and Mell [CM10] indicate.

Typical for cloud offerings is that certain tasks are managed by the provider. This is one of the reasons for using clouds after all. However, this also forgoes control of one's own data to a certain extent and renders specific forms of access management ineffective. This might be the case when enforcement of access policies is not based on encryption and a cloud provider contracts a third party to back up data, as noted by Bisong, Syed, and Rahman [BSR11]. On such a backup, access privileges are not enforced and unexpected or unauthorized restoration might expose it to the public.

Architecture

Including cloud offerings into own applications has many implications and consequences due to the architecture of such services. They represent externally owned and remotely managed instances, where a user's influence greatly varies depending on the specific service and offering.

For one, there is the general field of web technology security. Dawoud, Takouna, and Meinel [DTM10] point out that since clouds are exposed to the web and communicate using common web protocols and APIs, they naturally inherit all vulnerabilities and threats found in these technologies. Together with Almorsy, Grundy, and Müller [AGM16], they highlight widely used and adopted technologies, such as Simple Object Access Protocol (SOAP), Remote Procedure Call (RPC) protocols, File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP) and their implementations to be a common source of vulnerabilities. Methods used in penetration testing or cyber-related criminal activities, such as port scanning, can be used by an adversary to further identify vulnerabilities in web services. Threats can target a service or customer directly, intercept communication or forge sessions, ultimately leading to leakage of data. Common threats include Denial-of-Service (DoS) attacks, where a service is rendered unusable due to overloading its resources, or MITM attacks, where a data transfer between two parties is intercepted and potentially even altered, as explained by Williams [Wil14]. Attacks, such as Domain Name System (DNS) cache poisoning, can be carried out when a cloud provider does not implement necessary protection or is using misconfigured infrastructure. Various attacks target vulnerabilities in Extensible Markup Language (XML) parsers and validation of transferred data and data structures in general.

Attacks, trying to exploit a cloud provider's data validation implementations by injecting manipulated input as well as exploiting billing, are potential threats from other customers, as Grobauer, Walloschek, and Stocker [GWS11] describe. Therefore, it is only natural that cloud providers seek to protect themselves and only expose a small set of necessary controls to their customers. However, this course of action also strips customers of ways to ensure security of their used services and data through monitoring. With only little mechanisms for auditing available, it is not possible to set up proper means to detect security breaches or intrusion attacks. This makes risk assessment complicated or even impossible, as explained by Jansen [Jan11], who further mentions the absence of standardized security metrics for clouds. Even if customers can perform audits, the cloud provider has to respond to findings and patch identified issues. As a consequence, a client has to rely on their cloud provider's proficiency in secure infrastructures. The provider is responsible for taking necessary auditory measures, notifying their customers in the event of security intrusions or other disruptions, mitigating attacks and timely fixing corresponding problems. Grobauer, Walloschek, and Stocker [GWS11] and [HRFF13] illustrate scenarios where a cloud provider fails the security expectations of its customers by employing deprecated cryptographic ciphers, leading to exploitable vulnerabilities. Furthermore, a cloud provider might not even take security measures for internal data transfers or storage, making customers vulnerable to attacks from adversaries, such as other customers.

Other than unknowingly risking their customers' security, certain security-impairing decisions are taken deliberately. As described by Jansen [Jan11], providers not only share hardware among customers in Infrastructure as a Service (IaaS) offerings, *multi-tenancy* is often used in Software as a Service (SaaS) offerings. While this allows a provider to effectively use available resources and operate more economically, it also means that multiple customers get served by a single instance of a software that handles each customer's data together. Multiple customers' data can arbitrarily be co-located, as pointed out by Grobauer, Walloschek, and Stocker [GWS11]. Data isolation is difficult or not even possible in such setups and a customer is often not aware of this happening. A user has typically no control over the internal configuration and architecture of a provider's service offerings and can, therefore, not influence how his data will be handled. This lack of control creates further problems. Even when customers implement security measures within their own software on Platform as a Service (PaaS) or IaaS solutions, a provider can perform operations they do not anticipate. For example, a provider can persist data at any time as part of their internal management, and with that copy a customer's data before it was encrypted. Likewise, customers can never be sure if their data is really deleted as they have only little control and often no knowledge of their provider's internal processes, such as backups or other parts of redundancy and availability guarantees.

In summary, not only do we rely on a provider's competence, trust is required as well. In the worst case, a provider is an adversary or forced to act like one. The possibilities are endless: stored data, even provided libraries and runtime environments can be manipulated, therefore, even own code cannot be trusted. Threats beyond a customer's data are possible when a provider possesses further sensitive details about a customer, such as billing information.

The nature of clouds offering ubiquitous, distributed services can also cause non-technical concerns because data is subject to the jurisdiction of the country the servers are placed in, as Bisong, Syed, and Rahman [BSR11] point out. A country can impose restrictions on the strength of encryption algorithms and might be legally entitled to access a customer's data.

Virtualization

One of the reasons so many attractive cloud offers exist today are advanced virtualization techniques. They made it possible to easily manage and control large clusters of hardware, even heterogeneous environments. Spinning up new Virtual Machines (VMs) or tearing down running instances is done within seconds with a single click in a controlling instance called Virtual Machine Monitor/Hypervisor (VMM). User management is equally straightforward and some of this control can even be exposed to users, allowing them a certain amount of self-management, while isolation of VMs is maintained by the VMM. The ability to only scale up when necessary and otherwise use minimal resources while only paying what is really used, is a great advantage for customers that otherwise have to support and maintain large amounts of server-power themselves.

Having said that, virtualization also introduces a whole new layer of potential vulnerabilities. Both VMM and VM can contain vulnerabilities that allow an attacker to break isolation and manipulate internal processes as well as other VMs, as stated by Mather, Kumaraswamy, and Latif [MKL09].

Fundamental features that make virtualization easy to manage are also targets of threats. Dawoud, Takouna, and Meinel [DTM10] and Garfinkel and Rosenblum [GR05] show how the migration of a VM — moving it from one physical host to another — can be attacked by either intercepting migration data during transfer or by redirecting the transferred VM image to a malicious destination. Not only can this lead to an adversary gaining illegitimate access to data, such a migration can also move malicious VM images behind firewalls and other security measures, into trusted security perimeters, where they can do their harm. As a VM image is simply a file at its base, modifications and infections with malware are as easy as changing the image file. Equally, a simple *copy* operation can be considered theft. This implies that a VM image is vulnerable to threats even when offline and needs additional protection.

Hashizume et al. [HRFF13] describe further threats to data confidentiality. Snapshots of customer VMs might be made by a cloud provider as part of their availability guarantees. They usually contain the full state of a VM, including their data. Not only is this process often transparent to the customers, the location and protection of these snapshots is as well.

Snapshots are used for rollback of a VM, or more generally restoring a consistent state, potentially even into the future, and Garfinkel and Rosenblum [GR05] see multiple problems here. The security of a VM changes over time as vulnerabilities are introduced and fixed, malware and viruses are caught and removed. If an adversary can trigger a rollback, either through a vulnerability in a cloud provider's internal management mechanisms or with help from the provider, a less secure state can be restored. Known vulnerabilities of this state can then be used to threaten data confidentiality and integrity. A rollback to an earlier state can also influence or outright compromise cryptographic protocols. These often rely on cryptographically secure pseudo-random number generators. Using snapshots and rollbacks, one can revert the VM to a point where a random number was already determined but not yet used. This weakens several protocols, for example, zero-knowledge based protocols, which leak private data when the same random number is used multiple times. In the worst case, sensitive data, such as cryptographic keys, are persisted during logging or in a snapshot and can be accessed by others.

Grobauer, Walloschek, and Stocker [GWS11] further identify public repositories for machine images as vulnerability. Many cloud providers of IaaS or PaaS offerings provide such repositories, often as marketplaces where third party companies can offer supplementary services. Such images can already contain malicious code or be otherwise prepared in order to allow adversaries to access or leak data. Moreover, once an image is released and in use, it is difficult to provide software patches for detected vulnerabilities. Garfinkel and Rosenblum [GR05] add that patch management for VMs is a complex problem as their state is not linear. Instead, it is more akin to a tree where multiple instances with different patch-level and inhomogeneous software versions can exist at the same time. In the end, detected problems can remain unfixed for long and potentially even resurface at a later date, when a VM was rolled back. Since VMs cycle between online and offline states, infections can survive defensive and cleaning measurements and remain dormant until an infected VM is activated again in the future. Because VMs also scale up and down rapidly as instances are stopped, paused or new ones are created all the time, vulnerabilities and other security issues get additionally amplified. This highly dynamic nature of VMs also makes it hard to uniquely identify them. Commonly used values, such as a Media Access Control (MAC) address, can be changed on demand as needed.

An adversary can also try to actively deprive us of available resources to disrupt operations, as described by Dawoud, Takouna, and Meinel [DTM10] and Hashizume et al. [HRFF13]. Ristenpart et al. [RTSS09] adds that this can also be used to create so called *covert channels*. These are means to gain information about a target in ways that were not intended as communication channels, for example, by observing the system load of certain processes or VM instances. Furthermore, processor components, such as cache, pipelines, floating-point multiplier and branch predictors, as well as the memory bus, were mentioned as potential covert channels. Zhang et al. [ZJRR12] specifically describe how a malicious VM can extract cryptographic encryption keys from another VM. This is accomplished with a cache-based covert channel: by repeatedly filling the processor's cache and measuring the cache load times, one can make predictions regarding the computations done in between. Such timing information can be used to infer knowledge about a victim's cryptographic key, when the used encryption system's operations have execution times dependent on the key's value. Such an attack, as well as many kinds of covert channels, require an attacker to be located on the same physical hardware. How this can be accomplished is shown in detail by Ristenpart et al. [RTSS09], demonstrating their approach on Amazon's Elastic Compute Cloud (EC2). They successfully map the internal infrastructure of Amazon's IaaS offering and use this information to instantiate a VM on the same physical hardware as their victim, enabling them to carry out attacks. Monitoring a shared network enables attacks, such as a keystroke timing attack. It allows to significantly speed up searching for a password by measuring the timing between a victim's individual keystrokes while entering a password. This is possible because cloud providers often employ insecure virtual networks to share their network hardware between customers, as mentioned by Wu et al. [WDWY10].

Distributed Processing

While cloud offerings allow for parallel processing in shared-memory fashion, they also make it particularly easy to distribute processing across multiple compute nodes. Clouds, with their ability to dynamically adjust to their customer's needs as they release or request more instances, are a suitable fit for such scenarios. As a cloud does not incur constant cost like a traditional, self-owned data center does, there are ambitions to transitions towards the usage of multiple clouds at the same time, further promoting distributed processing of data.

AlZain et al. [APST12] present several benefits that utilizing multiple clouds in conjunction can have. Even though cloud providers go to great lengths to achieve high availability rates, constant availability is not realistic. For customers, using multiple clouds can further increase availability and redundancy. Additionally, customers can easier protect themselves from adversaries among employees of a single cloud provider. In the event of a cloud provider going out of business, a multi-cloud environment can cope with only minimal interruptions.

However, multiple clouds also raise concerns. Previously, customers sent data to their cloud and received results, with additional data transports within the provider's infrastructure. With multi-cloud environments, data transfers between two providers are also possible. Customers have little control over it and only few ways to assess the security measures taken to ensure confidentiality and integrity [Jan11]. Cloud providers and offerings differ vastly in their feature set. While the basic set of functionality is usually aligned, protocols for access, user management and hardware capabilities are inhomogeneous. For this reason, systems and frameworks that abstract multiple clouds are used. Fortunately, problems, such as Byzantine faults, are within the scope of such systems and do not need to be considered here. Still, existing systems and distributed processing frameworks, such as Apache Hadoop, contain vulnerabilities. In case of Apache Hadoop, authentication using Kerberos was added [DORZ11] with limited support for fine-grained permissions and authorization, based on unencrypted data. Integrity or privacy for network communications is not provided and query processing is only supported for unencrypted data.

3.2 Solutions

As preventing all threats and eliminating all vulnerabilities is an immense task, multiple measures have to be taken together. Most research focuses on one particular aspect of cloud computing, sometimes assuming preconditions that are often not feasible in reality. Zissis and Lekkas [ZL12] identify key technologies and components, such as cryptography, Public Key Infrastructure (PKI), authentication and authorization techniques, that, chained together, protect confidentiality and integrity of a customer's data. However, the security of a system not only depends on the technologies being used but also how they are applied. Vulnerabilities can be created by flaws in programming and conceptual weaknesses in the security architecture alike. It is also important to consider the use case, as it is not trivial to port security solutions from one scenario to another. Different requirements often call for different approaches. Only few holistic solutions exist and these have to be carefully evaluated regarding their assumptions, use cases and security guarantees. Finally, among all security concerns, many problems cannot be tackled by a customer of cloud offerings alone but require collaboration with a provider.

In the following, research tackling the mentioned security concerns is discussed and evaluated regarding their approaches and potential shortcomings.

Data Security

In order to achieve secure storage, Kamara and Lauter [KL10] propose a system built on top of a semi-trusted service provider. Within this additional layer, data is encrypted with a symmetric scheme. The respective symmetric keys are encrypted using Ciphertext-Policy Attribute-Based Encryption (CP-ABE), a public-key encryption scheme where a set of *attributes* is attached to every user's key and an access policy is associated with each ciphertext [BSW07]. Using CP-ABE, users can only successfully decrypt a given ciphertext when the policies associated with the ciphertext match their key's attributes. An index is created over the data files and encrypted using a searchable encryption scheme. This is a trade-off between functionality and security as such schemes suffer from leakage of metadata, for example, what files have a certain keyword in common. The use case assumes multiple users want to share data securely with each other and host it on the same cloud provider. For a user A to share data with user B, active participation is required by A in form of generating a token and credentials that can be used to query data and decrypt the result set, respectively. Special, uncommon cryptosystems have to be used and pose fixed requirements on the hardware and features of the cloud offering being used. Furthermore, the authors stress that their concept is an illustration on how certain cryptographic schemes could be used together and not a finished system.

A proposal by Waizenegger [Wai17] allows secure deletion of data in an OpenStack object store ("Swift"). Users encrypt their data before uploading to the cloud. There, the encrypted data is stored alongside with unencrypted metadata. Deletion uses so called cryptographic erasure, where simply the key for encrypted data is deleted. As decryption becomes impossible, the data in question can no longer be read, assuming the employed encryption scheme will not be compromised in the future. Due to data being encrypted on the client-side, a malicious cloud provider can only read metadata and does not need to be trusted regarding successful deletion. As a storage service, it can easily be added to any existing solution in need of a data store with secure deletion.

Another work in the area of secure deletion of data, this time specifically for backups, is proposed by Rahumed et al. [RCT+11]. The integral feature is support for versioned incremental backups where one file can be part of multiple versions. When a file is encrypted with key k_{file}, this key itself is then encrypted once for each version this file appears in, creating a key k_{vi} corresponding to the respective version. Deletion removes version key k_{vi} and a file remains accessible as long as at least one key k_{vi} exists that is capable of decrypting k_{file}. This can be considered as a kind of reference counting. As this is intended to be a layer on top of an existing storage solution, it can easily be added wherever a need for a data store with secure deletion for versioned backups arises.

Aggarwal et al. [ABG+05] present a privacy-preserving, distributed database architecture. Encryption is not required and the cloud does not need to be trusted. This is accomplished by splitting the data and storing it on two or more providers. The data is split so that one compromised provider is not enough to violate privacy. Queries from users have to be pre-processed and sub-queries for each of the involved providers have to be created. Likewise, the result sets have to be post-processed and split data has to be merged. However, several assumptions are made. For one, untrusted providers are not allowed to communicate with each other as their collaboration could undo the data splitting. Moreover, it is assumed an attacker cannot eavesdrop on more than one communication channel with the providers because no communication security is employed. Finally, confidentiality is not preserved, only privacy. The assumption is that, for example, leaking both the name and social security numer of a customer conjoined is a problem but leaking this information separately, without relationship, does not violate privacy. As database service, it can easily be used like other services and integrated into existing systems.

To securely store sensitive data, such as cryptographic keys on multiple servers, Damgård et al. [DJNP13] propose a scheme that makes use of spreading keys across multiple servers. The scenario for this proposal is well-defined: multiple servers that perform long-lived computations for multiple parties and are subjected to cycles of online and offline (idle) periods. Computations are carried out during online periods and the involved parties do not wish to share their secrets with each other, only the computed result. This is called (Secure) Multi-Party Computation (MPC) and useful for tasks such as electronic elections and electronic auctions. One exemplary instance of a MPC problem is the so called *Millionaires' problem*, where two millionaires want to figure out who of them is richer, while hiding their wealth from each other. Appropriate protocols for MPC exist and can be employed during online periods of the servers. The main contribution here seeks to protect data during offline periods in such setups. During these, data is encrypted to protect confidentiality at-rest and the encryption keys are distributed among other involved servers, so that a minimal number of servers has to be compromised before the key can be restored. It is assumed that servers act fully autonomous over a long time. In case this is not applicable and servers are not autonomous, the security guarantees are weaker. While this effectively solves the general problem of storing cryptographic keys alongside the data they encrypt, it requires multiple long-lived servers to be used conjointly and a workload that fits a cyclic online/offline pattern.

Access Management

Yan, Rong, and Zhao [YRZ09] present an approach to identity management using hierarchical identity-based cryptography in a multi-cloud environment. Identity-based cryptography is a public key cryptosystem where a publicly known string value is used to generate a public key for a user, for example, the user's email address. This is done using a master public key and a corresponding secret key is generated using a master secret key by a trusted third party. Aside from this trusted third party acting as secret key generator, there are sub key generators per cloud, thus, making it hierarchical. This simplifies key management in scenarios where multiple companies use diverse multi-cloud environments together. Because everybody has access to everybody's public keys, all users can mutually authenticate each other. However, identity-based cryptography has shortcomings regarding key revocation. In order to revoke a user's key, the master public key has to be changed. This requires a regeneration of all valid public keys. Alternatively, the revoked user's identity has to be chanced, which can be rather difficult in practice, when email addresses or phone numbers are used. Moreover, a key escrow situation exists. As the third party holds all users' private keys, it has to perform all tasks — such as signing — that involve a secret key on behalf of the respective user. Lastly, the third party must be absolutely trusted.

Because key management systems are all about not storing keys where they are used, Lei, Zishan, and Jindi [LZJ10] propose a key management infrastructure consisting of a dedicated client and server. The server is composed of a symmetric key management system and a PKI as backend. Client and server communicate using an own protocol. The major difference to open standards, such as the Key Management Interoperability Protocol (KMIP)[1] by the Organization for the Advancement of Structured Information Standards (OASIS), is supposed to be interoperability between clouds as different clouds can communicate and exchange information about keys. However, for this to work, cloud providers would be required to integrate this protocol and support this proposed infrastructure at a large scale.

Liang et al. [LAL+14] propose a modified version of proxy re-encryption (cf. Section 2.2.1) called *DFA-Based Functional Proxy Re-Encryption*, allowing for better access management. Pieces of additional data are appended to encrypted files and Deterministic Finite Automata (DFA) are attached to cryptographic keys. Decryption is only possible when this additional piece of data is accepted by a key's DFA. A semi-trusted proxy performing re-encryption can alter the attached piece of data along the transformation of the ciphertext. In settings where Proxy Re-Encryption (PRE) is considered for access management, this adds more freedom to express complex policies. Compared to CP-ABE

[1]Communication protocol for manipulating cryptographic keys [Org17b]

PRE, which adds a set of attributes to private keys and access policies to ciphertexts, this approach is secure against Chosen-Ciphertext Attacks (CCAs).

A system by Tang et al. [TLLP12] allows access management for storage services. It is designed to be an overlay over an untrusted provider and works on top of a cloud storage solution. Data is encrypted with a symmetric cipher. The respective keys are encrypted using an asymmetric cipher with keys containing an access policy. These keys are managed by a quorum of *key managers* that operate using threshold secret sharing and, therefore, are considered trusted. Furthermore, CP-ABE is used for communication between client and key manager to enforce fine-grained access policies. The system supports timed expiration and assured deletion by revoking file access. Potential use cases include backups and file-sharing using untrusted storage providers. A prototype running on Amazon S3 was tested and showed a factor of increased estimated storage and transfer costs of less than two, for a series of `PUT` and `GET` requests.

Another solution for fine-grained access management is proposed by Wang, Liu, and Wu [WLW10]. They propose *hierarchical attribute-based encryption*, a combination of hierarchical identity-based cryptography and CP-ABE. In such a setup, revocation is difficult because whenever a user associated with an attribute a is revoked, all keys with the same attribute a have to be updated. In order to postpone these updates, make updating of keys lazy, PRE is employed. A version number is used for each attribute used in CP-ABE and increased when an associated user is revoked. When other users later request access to a file associated with a, the version of their attributes is checked. In case an older version is found, the users are told to upgrade their keys and the system will re-encrypt the requested data so it can be accessed using the updated keys. This is supposed to help enterprises share parts of confidential data.

Chow et al. [CCH+12] present a dynamic system that preserves confidentiality of data in the cloud with access management for users. It is dynamic with regard to new users as they can join the system at any given time and are still able to decrypt the encrypted data. This is accomplished by a modified broadcast encryption scheme. Broadcast encryption allows a ciphertext to be encrypted once and decrypted by a group of users. As the name suggests, broadcast encryption originates from broadcasting channels and is also employed for the Content Scramble System (CSS) used for DVD encryption and the Advanced Access Content System (AACS) used for Blu-ray disc encryption. There are various different approaches to this problem, all with the requirement of having a dynamic set of users, changing on every transmission. Additionally, simple revocation of users must be possible without affecting other users. Access management is accomplished by assigning data files to classes and specifying who can access which classes. The system is CCA secure and does not incur notable storage overhead. Through the usage of group signature schemes, users can act anonymously while accountability is still maintained because their identity can be revealed. The system's use case and dynamic user management is specifically tailored to data sharing among a large group of users that is constantly changing.

Architecture

In order to tackle this diverse set of threats and vulnerabilities that arise from the architectural circumstances of cloud computing, Almorsy, Grundy, and Müller [AGM16] suggest an equally diverse set of solutions and mitigation strategies. Among others, secure VM repositories, improved VM isolation and secure virtual networks are mentioned. Secure APIs with proper isolation, open standards in authentication and authorization as well as protection from attacks targeting specific parts of the Service-Oriented Architecture (SOA) — such as XML-related attacks, DoS, MITM — are described. Web-facing components have to be scanned for known vulnerabilities and protocols, such as SOAP have to be secured, so they will not leak data or allow other tampering by unauthorized users. Finally, intelligent key management and proper identification are required so that encryption can successfully protect confidentiality of data. Jansen [Jan11] additionally recommends conventional client-side and server-side protection, using techniques such as operating system hardening, in order to prevent a malicious client from compromising a server or service.

Furthermore, Dawoud, Takouna, and Meinel [DTM10] consider physical vulnerabilities, such as malicious employees. Locked high security rooms with physical access management and monitoring prevent adversaries from damaging hardware, in order to deny services to a customer, theft as well as corruption of data due to direct physical access. However, such security measurements are problematic, as a customer has to take a provider's word for it and cannot verify if such mechanisms are really enforced.

Wang et al. [WRLL10] explain the importance of audits for the security of cloud storage. Detecting data corruptions as late as when data is accessed is undesirable and potentially too late for graceful recovery. Further, they advise that audits, where data has to be downloaded in order to be verified, are impractical. Therefore, Wang et al. [WCW+13] propose a protocol to make public auditing on secure storage possible, while maintaining privacy. Public auditing refers to the ability of an external party, for example, a contracted company, to audit cloud storage without the need for downloading the content in order to verify it. Therefore, confidentiality and privacy is protected. Before a client uploads data to the cloud, it is necessary to compute a Message Authentication Code (MAC) for at least a subset of files and send these codes to the auditor. A MAC allows to verify the integrity of the file as well as the authenticity of its creator. For the same purpose, a Homomorphic Linear Authenticator (HLA) can be used. It allows to be aggregated and computed over a linear sequence of data blocks of arbitrary length. To perform an audit, the auditor asks the cloud provider to compute such MACs for files stored online and compares them to their local copy they previously received from the client. The protocol emphasizes efficiency for the third party auditor and allows batch processing of multiple clients. Non-invasive auditing of such form can be added on top of an existing system. The use case considers situations of data storage, where other means of integrity verification or encryption are considered overkill, too expensive or key management too complicated.

With their system *SecCloud*, Wei et al. [WZC+10] consider more than only secure storage. Additionally, they consider a malicious cloud provider that might either manipulate computations or only pretend to have performed them. The system at its base uses audits, essentially sampling in intervals. Data integrity is verified using data signing and auditing of computational results is accomplished by verifying a probabilistic subset of all computations. This requires a trusted third party as auditor which is expected to have significant computational resources. The cloud needs to implement the devised protocol to communicate with this trusted auditor. As an auditing scheme, it can be added on top of an existing system that uses secure storage and computations and wishes to increase security. However, probabilistic auditing alone does not make computations fully secure, it only increases the chances to detect potential fraud.

Vieira et al. [VSWW10] propose an intrusion detection system for environments with many compute nodes. On each node, extensive logging of events is performed and analyzed. Based on logs, they try to identify and trace back potential attacks to the responsible user. User behavior is analyzed in order to preemptively find suspects. The system mostly acts reactive to situations that have already happened, preventive and protective actions are limited to malicious behavior that is correctly anticipated. Such a monitoring system could be combined with existing processing systems, given that their middleware can be deployed and is able to access the necessary information.

As a holistic solution for security in the cloud, Santos, Gummadi, and Rodrigues [SGR09] present their Trusted Cloud Computing Platform (TCCP) because customers using a cloud offering, even in case of IaaS solutions, cannot protect themselves against a malicious provider or insiders with elevated access. Therefore, this proposal suggests a closed box execution environment that cannot be tampered with, even by the cloud provider's system administrators. This is accomplished by running a so called Trusted Virtual Machine Monitor (TVMM) on every node. It manages and hosts the VMs of customers as well as restricts access even for privileged administrators. Further, it maintains and protects its own integrity. Similar systems protect single VMs and are, therefore, unable to ensure security beyond it. For this reason, a so called *Trusted Coordinator* is employed. It manages a set of TVMM-running nodes per customer, essentially establishing a confined, secure and trusted space. Coordinator and TVMM collaborate to protect VM images even during migration and prevent them being started on untrusted hardware. To make all of this possible, a special hardware chip is required and used to install the TVMM in a secure boot process on the respective node. A client is able to verify remotely if an implementation of TCCP is running. All in all, a cloud provider needs to implement the necessary software as well as hardware to run this system and then expose it to its customers.

Virtualization

As Garfinkel and Rosenblum [GR05] see the problems of virtualization in the flexibility and heterogeneity of VMs, their evident solution to security problems is to strip them of this versatility and impose severe limits on functionality. This is accomplished by using a virtualization layer that will take over security and management functions, usually found within the guest operating system of a VM. Furthermore, policies enforce limits on VMs and can grant exceptions for certain users. This proposal is only a concept and would need to be implemented by a trusted cloud provider.

With their *Advanced Cloud Protection System*, Lombardi and Pietro [LP11] propose an approach that extends protection of VMs to the other cloud infrastructure components involved. The system is based on extensions for the Kernel-based Virtual Machine (KVM), which is based on the Linux kernel and essentially turns it into a VMM. Instead of performing emulation itself, it rather exposes an interface for a host to be used. To achieve its goal, the proposed system transparently monitors key components of the cloud architecture and potential targets of an attack. Active as well as passive monitoring is used to guarantee integrity of both, middleware and kernel. Entry points into the system are monitored by logging and periodic verification of used binaries and libraries. The protection provided by this system covers attacks from other VMs as well as attacks from outside the cloud. It prevents attacks on VMs that might result in the leakage of data, improves data segregation between customers, and is fully implemented. However, to use it, a cloud provider has to deploy and support it. It also assumes the cloud provider is a fully trusted entity and considers a VM image to have integrity up until it is exposed to the network.

ZeroVM from Rad et al. [RLP+14] is a different approach to virtualization in the cloud. Typically, data is either uploaded by a user or moved from a data store to a VM in order to process it by an application. With ZeroVM, the application is moved to the data store to process the data directly where it is located. It is designed as middleware package for an OpenStack object store (Swift) and executes *ZeroVM Application Packages*. Security is gained by processing data right where it is stored and, therefore, reducing data movement. Other than that, it claims deterministic processing execution, increased isolation and provides a lightweight container-based virtualization platform. It has to be implemented by a cloud provider that needs to be trusted. Additionally, data processing software needs to be written for, or ported to the Google Native Client (NaCL) platform in order to use it as ZeroVM Application Package. However, other than rewritten or ported software, it also requires redesigned processing patterns and workflows to be compatible with this different approach to data processing in the cloud.

Instead of changing current paradigms of data processing in the cloud, Szefer et al. [SKLR11] left VMs where they are. However, identifying the VMM as a major source of vulnerabilities, they conclude to remove it in a system called *NoHype*. VMs now natively run in parallel on the underlying hardware. An interruption in a VM that gives control back to the VMM, could get exploited by the guest but is now no longer possible. Modifications to the operating system are required for this system to work, making it rather invasive. Moreover, so called *hypercalls*, required by paravirtualization to request privileged operations, are not supported. Completely removing a system containing vulnerabilities is a reliable way to eliminate them, however, new ones will be inevitably introduced along the way. NoHype does not secure VM related mechanisms, such as migration or snapshots and it can only be implemented by the provider of a cloud, whom we have to trust.

Wu et al. [WDWY10] present a framework for virtual networks. In virtual networks, multiple clients share the same network hardware. This makes various attacks possible that result in data leakage, such as sniffing or spoofing attacks. Not just privacy is endangered, as careful monitoring can leave an adversary with the SSH password of its victim through keystroke timing attacks. The proposed framework consists of three layers (routing, firewall, shared network) and provides stronger isolation. As with every other solution around the topic of virtualization, the cloud provider needs to implement these approaches and systems.

Distributed Processing

Data can be reliably protecting while in-transit and at-rest using cryptosystems and message authentication. However, to process data it usually has to be decrypted, exposing a weak point attackers can target. Proper access management can help to restrict the audience of sensitive computations as well as the corresponding results but does not keep data strictly confidential while in-use. Homomorphic encryption (cf. Section 2.2.2) is one kind of encryption that allows to perform computations on encrypted data, omitting this weak point. Due to the high complexity and computational resources needed, Naehrig, Lauter, and Vaikuntanathan [NLV11] attested it a limited use for general purpose systems. They present specialized use cases, where the application of a partially homomorphic cryptosystem is viable, such as contextual advertisement that involves private information about the targeted users. Nevertheless, homomorphic encryption does not solve all problems, as it does not automatically guarantee that the computations performed are the computations desired. Due to it being malleable, an adversary can still attack computations using homomorphic functions and manipulate results by, for example, subtracting values that should not have been.

Bacon et al. [BEP+14] evaluate possibilities to apply Information Flow Control (IFC) to the cloud to achieve secure computing. In this data-centric approach, access policies are attached to data in form of security labels and define which user has what kind of access. A special runtime environment tracks all data in the system and limits data propagation by enforcing these access policies. Alternatively, a static analysis of programs can be conducted to ensure their data usage complies with the policies. Depending on the choice of labeling mechanism and scheme, complex access hierarchies can be modeled. This is not only confined to the user-space inside a VM but can include the VMM and surrounding environment as well, depending on what level the IFC is implemented. Therefore, it also provides isolation in multi-tenant environments, where multiple clients with different access privileges share a single environment, as they are often used in SaaS offerings. It also allows to secure data processing without the need for encryption, when the system can ensure that no data is leaked to unauthorized users. A provider can expose this functionality and clients can attach security labels to their own data prior to upload. However, there is no way for a client to ensure that access policies are correctly enforced. In case of IaaS offerings, clients can use their own IFC-enabled operating system but are still subject to vulnerabilities of components this operating system runs on top of. Additionally, the tracking of data often assumes a benevolent developer who does not deliberately try to negatively influence tracking. Moreover, it is not guaranteed that all data flows are tracked and some data flows, such as covert channels, are inherently not tracked.

To abstract multiple clouds, Bessani et al. [BCQ+13] propose a system called *DepSky*. It exposes a secure, distributed data storage featuring various improvements due to it building a *cloud-of-clouds*. Increased availability is achieved by replication across multiple clouds as well as the usage of erasure codes. Additionally, this enables corruption protection and prevention of vendor lock-in. Encryption is used to address confidentiality and keys are managed with a secret sharing scheme, therefore, spitting keys across clouds, with a quorum required to access a key. However, the system only considers storage and does not consider processing or any kind of code execution in its security concept. The features depend on the usage of multiple clouds and the redundancy, which will inevitably increase costs and might therefore not be desirable. Moreover, the secret sharing scheme used is incompatible with most other access or key management approaches.

Apache Hadoop is a mature framework for distributed processing and storage. Nonetheless, it lacks various security related functionality. This situation improved with authentication using Kerberos, extensions for access tokens and delegation of privileges as proposed by Das et al. [DORZ11].

In recent versions, Hadoop offers optional encryption using special directories within the Hadoop Distributed File System (HDFS) [Apa17]. This encryption is end-to-end between clients and transparent, meaning that no explicit encrypt or decrypt action has to be issued by a client, similar to Full Disk Encryption (FDE). This implies that data in-use is not encrypted. It provides security for data at-rest and in-transit within the HDFS. Data is organized in *encryption zones*. Multiple zones can exist and each one is associated with its own unique encryption key. Files within an encryption zone are encrypted using per-file keys. These keys are encrypted using the closest zone's key and stored in encrypted form alongside the file they belong to. Access control for encrypted files is managed using regular HDFS file system permissions. However, an attacker taking over a privileged account can only access encrypted data and the corresponding encrypted keys. Decryption keys are managed by a separate entity, a key management server, which is additionally enforcing an own set of permissions.

In cases where stronger encryption is preferred, Yang, Lin, and Liu [YLL13] propose a triple encryption scheme for HDFS. Files are encrypted using a symmetric cryptosystem and the respective key is encrypted with an asymmetrical cryptosystem, using keys of the respective owner of the file. This is called a hybrid encryption scheme. A data management system associates files with users to keep track of the required keys. This hybrid encryption scheme is extended by another layer, utilizing the International Data Encryption Algorithm (IDEA) to encrypt a user's asymmetrical keys. That makes it possible to store them on the server as well, sparing the user from key management tasks and lowering the risk of keys being stolen from insecure users. Aside from encrypting data at-rest, processing of data is not considered. Access management is based on users and, therefore, does not trivially support file sharing with others. Multiple users with access to the same file implies them to either share their keys — and with that share all data encrypted with these keys — or have multiple versions of the same file, each encrypted with different keys. The former makes key management complex, in case of many users sharing files with different audiences.

Lin et al. [LSTL12] propose one more hybrid encryption scheme for HDFS. Files are encrypted with a symmetric scheme and random session keys. These keys are encrypted using an asymmetric scheme and keypairs per owner of a file. One noteworthy detail about their implementation of symmetric file encryption is the combination of block cipher and stream cipher for the last part of a file, to avoid wasting space due to padding. Compared to Tahoe-LAFS — a secure and encrypted distributed file system that can be used by HDFS clients with an adapter — this proposal uses fewer keys (per user), resulting in easier key management. Tahoe-LAFS uses a key per file, resulting in a more complex key management but also allows better fine-grained permissions. This proposal stores a file's symmetric key as its header, resulting in a linear storage overhead on the servers and a constant overhead for users in form of their asymmetric key. Tahoe-LAFS does not exhibit a storage overhead on the servers, instead users have to store these keys themselves. The implementation was tested and is within 50–75% read performance of unmodified HDFS, with Tahoe-LAFS being slower. For writing performance it behaves vice versa.

G-Hadoop, an extension to Hadoop making MapReduce tasks across multiple clusters possible, is further extended with a security model by Zhao et al. [ZWT+14]. G-Hadoop replaces HDFS with Gfarm, an open source, petascale grid file system. The designed security model consists of multiple components, such as Single Sign-On (SSO) for authentication, asymmetric encryption for communication, access control, and protection in form of detection of malicious users. Access control is fine-grained with respect to the involved components, so that individual users can have no access to certain slave nodes. This is possible because access privileges of users are stored in the user database for every single node. The security model ensures privacy so that slave nodes do not see user information. To send jobs, users authenticate to the master node using SSO. Subsequently, a temporary user instance will be created and used to process the job. Proxy credentials with limited lifetime are created for this temporary user in order to authenticate itself with slave nodes. The framework reliably prevents a series of common threats, such as MITM and replay attacks. Proven technologies, such as Secure Sockets Layer/Transport Layer Security (SSL/TLS), Secure Shell (SSH), and SSO are used in order to achieve security and no special cryptosystems are required. Internally, a Certificate Authority (CA) is used, adding to the rather complex setup. As with the other systems and frameworks for Hadoop, data processing is not strongly protected by means of encryption. However, in this case, at least access to nodes is limited to privileged users.

Hamlen et al. [HKKT10] conclude that a holistic solution to secure the cloud is too complex. Consequently, they propose a series of enhancements to Hadoop that, in total, are supposed to lead to a secure cloud. They also consider a setup composed of multiple clouds. Key features include secure and encrypted data storage, large scale query processing, fine-grained access control and strong authentication. As data format, only XML with RDF data is considered and used with an access control scheme that allows access to only a subset of a single document by utilizing view modification. Secure publication of data is done by splitting files and publishing them on untrusted third parties. Access policies are sent alongside the respective files and enforced by the third party. Because the party is untrusted, measures have to be taken in order to verify the authenticity and completeness of the files. This is accomplished by encrypting various compositions of files with the respective owner's key. To achieve encrypted storage, that integrates the published data, secure co-processors are used to handle encryption and decryption tasks. Secure co-processors are dedicated, verifiable and tamper-resistant hardware. Furthermore, VM and VMM are assumed to exist in a secure version alongside a secure virtual network monitor. They are used to provide isolation between users and data domains. For access management, they propose a privileged process that is capable of rewriting binaries of other programs. This privileged process will inline checks to enforce fine-grained eXtensible Access Control Markup Language (XACML) access policies into untrusted programs on the fly. This approach allows to perform history-based access policies that can be used to enforce fairness. With regard to secure query processing, solutions are investigated to enforce access policies during processing, for example, by inserting `WHERE` conditions into a query. The proposal covers many aspects of cloud computing. Secure co-processors help with reliable cryptography on untrusted providers and can store the most important keys. However, the assumptions and preconditions make deployment on a suitable cloud provider difficult, especially the secure co-processors and various secure versions of common software. The amount of access required to implement a process privileged enough to rewrite arbitrary binaries rules out SaaS and PaaS offerings and likely exceeds IaaS offerings as well. Therefore, to implement such an approach, cooperation with a cloud provider is inevitable.

Regarding the need to trust cloud providers, the question arises why tamper-resistant secure co-processors are not used to perform all relevant computations. Secure co-processors are very specialized, limited-purpose hardware. They are not equipped with vast computational resources and only simple software so that it can be completely verified [HKKT10]. *Trusted Computing* attempts to combine general purpose hardware with trusted software. This is accomplished based on a secure cryptoprocessor, such as the Trusted Platform Module [Tru17]. Combined with compatible software, a chain of trust can be established. Open source software does exist to construct such a chain, starting with a boot loader (TrustedGRUB), an operating system (Trusted Linux) and a VMM (Trusted Xen). Different possibilities exist to implement a Trusted Platform Module. While discrete chips provide a certain tamper resistance, implementations integrated within other chips or such that run in software are not required to be tamper-resistant. Consequently, achieved security and trust of the established chain can vary and might not be comparable to a secure co-processor. However, Trusted Computing is controversial because it delegates a high amount of control of hardware and software to third parties. Various critics exist, such as Richard Stallman [Sta17] (Free Software Foundation) and the Electronic Frontier Foundation [Ele17]. They oppose the amount of control it gives to big companies, allowing them to restrict software used by clients.

4 Security Concept

The objective is a generic concept that can be applied to many distributed data processing applications, in order to provide information security and protection from adversaries. It is intended to be used by users who wish to protect their cloud-based distributed processing, without forgoing the benefit of a vast market of cloud services, as it is available today. The result is a concept that protects its user actively by employing customized, protective actions. Additionally, recommendations are made that can increase security passively.

Following, the term *component* is used as an abstraction of a data source or dedicated processing step, be it a simple storage service or a distributed Hadoop program.

In this chapter, requirements are defined along with a threat model that has to be considered. Then, the proposed security concept is presented.

4.1 Requirements

The following requirements were derived from intensive literature research and considered relevant to a holistic security concept that is designed for customers of cloud solutions.

Req_1 **Access control for users** Users who commission processing of data should only be able to use sets of data and data processing routines they have necessary access privileges for.

Req_2 **Access control for processing components** All components handling data, such as servers or services in the cloud, should be subject to access control. Components should only be able to process data they are authorized to process.

Req_3 **Confidentiality** Eavesdropping or data leakage should not disclose plaintext data to adversaries. Confidentiality should be ensured for data in-transit and at-rest for every component involved that supports arbitrary but secure kinds of encryption.

Req_4 **Integrity** Data should be protected against modification by adversaries in-transit and at-rest, for every involved component and user supporting appropriate technologies and algorithms.

Req_5 **Identification** All users and components involved in planning and data processing should be uniquely identified.

Req_6 **Heterogeneous environment** Secure data processing should be possible with cloud solutions that are available today. Environments composed of many diverse configurations and capabilities should be supported and not restricted to a specific cloud computing service model.

Req_7 **Adjustable security requirements** As strong security and high performance are often diametrical requirements, it should be possible to influence the security concept regarding the algorithms it employs.

Req_8 **Flexibility** The security concept should not rely on a specific algorithm but rather be compatible with a wide range of possible technologies and algorithms. It should be able to incorporate future findings of cryptographic research.

Req_9 **Privacy** It should be possible to ensure special privacy requirements when needed, for example, when processing medical data with sensitive information.

Req_{10} **Scalability and distributed processing** A security concept should not limit scalability or distributed execution of data processing.

Ensuring *availability*, together with confidentiality and integrity often called the *CIA triad of security* [Win11], is considered a service provider's responsibility and part of the guarantees they make to their customers.

4.2 Thread Model

The threats and vulnerabilities a cloud user is exposed to are described in Section 3.1.

It is also necessary to know who threatens a cloud customer's security. The only trusted party is the user applying this security concept, along with components, in complete control of this user. The provider of a cloud service is not a trusted party, unless there are technologies available that allow users to monitor and verify the integrity of a provider and its services. Therefore, potential adversaries include untrusted providers, other customers of the same cloud service provider, and yet unknown parties alike.

4.3 Concept Overview

The approach of this security concept is guided by the strategy of Hamlen et al. [HKKT10] — chaining together a series of technologies — and utilizes key technologies loosely based on the ones identified by Zissis and Lekkas [ZL12]. A generic security concept for distributed data processing is a difficult task, mainly due to the high number of involved components, where each potentially carries a multitude of attack vectors. Besides, the degree of control over these components varies and is not uniform. Combining current cloud offerings for data processing results in a highly heterogeneous environment, with each component supporting a different set of features and levels of control. Additionally, security of distributed data processing systems in cloud environments does not start at the servers a cloud consists of. Rather than that, a flow of data has to be considered in its entirety. Starting with a user sending — or a source providing — data to the cloud, to a consumer receiving computations made online, all are part of the whole process and all of these involved parties have to be covered by a security concept.

In order to create a general concept that can cope with multiple use cases and constantly changing infrastructure, it is composed of two parts: a modeling phase where the processing steps are modeled as well as analyzed, and an execution phase where the actual processing takes place.
During modeling phase, it is algorithmically decided for a given data flow what measures to take in order to secure the upcoming data processing. This includes cryptographic ciphers, algorithms, protocols, solutions to access management, and additional or modified processing steps, if needed. User input can be used to further refine the parameters determined in this phase. This approach allows to find an optimal set of security measures for each distributed processing task, tailored to the individual user-needs, components and cloud services involved.
After modeling, in the execution phase, an arbitrary execution engine then orchestrates the specific data flow and takes measures to ensure security as previously determined.

4.4 Building Blocks

Two essential parts are involved in securing a data flow: the modeled data flow *plan* and a *security provider*, handling authentication, authorization and key management.

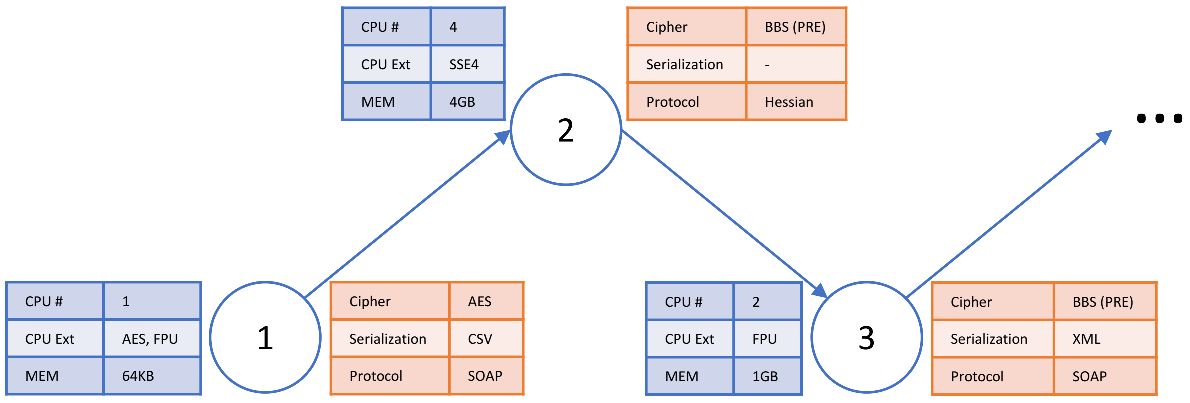


Figure 4.1: Annotated graph with components 1, 2 and 3. Component capabilities in blue and determined security parameters in orange tables.

4.4.1 Plan

A plan is an artifact of the modeling phase. It is based on a given directed acyclic graph that models the data flow of a user's distributed data process. Nodes represent components, therefore, processing steps, data sources and sinks. Directed edges mark the flow of data in form of transmissions. This graph is now attributed with information to secure the data flow, such as used protocols, additional processing steps, signature schemes and cryptographic ciphers to be applied along with parameters, such as encryption key length. These parameters are determined algorithmically for each data flow, based on the capabilities of available cloud environments and services involved.

Figure 4.1 illustrates a simplified example of an annotated graph. Hardware capabilities are listed in blue tables alongside their corresponding node. These values are used to determine the security parameters, represented as orange tables. Algorithmically determined parameter values are attached to nodes and can be logically divided into three distinct sets. Values relevant for:

1. incoming data transmissions: how is an incoming stream of data protected and formatted
2. outgoing data transmissions: how is an outgoing transmission of data protected and formatted
3. handling data: how to protect data on the current node, mainly data at-rest

This dynamic determination allows us to generate security parameters adapted specifically to each individual data flow's combination of cloud environments and services. It is important to notice that this approach is not limited to one specific cipher or cryptographic algorithm. Even major distinctions, such as asymmetric or symmetric

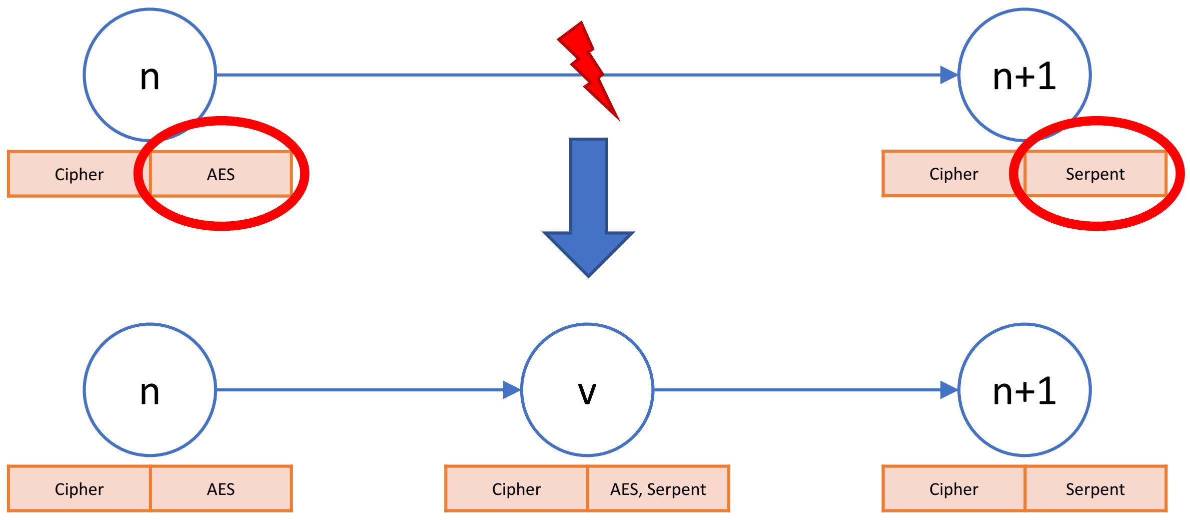


Figure 4.2: Component `v` is inserted to enable a secure data transmission from `n` to `n+1`.

encryption schemes, are not predetermined and can be selected based on the given situation. Choices, such as cryptographic algorithms, data exchange formats or communication protocols do not necessarily have to be identical for the whole data flow. Instead, these parameters can change between each component or user.

The resulting security parameters can also be optimized for different goals. For example, one can optimize for maximum security by selecting strong, cascade encryption approaches. In other cases, a small computational overhead can be accomplished by using only simpler, hardware accelerated ciphers, or even relying on a simple SSL/TLS connection because the target SaaS offering does not support anything else. Such modifications can be implemented by modifying the function `MostSecure` in Algorithm 4.1 to select, for example, the fastest instead of the most secure security parameters.

However, the input graph is not only annotated but also modified when deemed necessary. In case two connected components have an empty disjunction of parameters, when there is no cipher or protocol supported by both, additional processing steps can be inserted into the flow as an adapter to create compatibility. As illustrated in Figure 4.2, this is done by splitting the original edge and connecting both components with an additional component, creating compatibility and preventing an insecure connection. Such an adapter component can also be used to increase transmission security in cases, where two components have a broad range of supported security features but only share weak ones.

An implementation is depicted in Algorithm 4.1. For every edge $(n1, n2)$ of a given directed acyclic graph $G = (V, E)$, it first determines the security parameters supported by the connected nodes $n1$ and $n2$, based on their capabilities. If the intersection of both nodes' security parameters is not empty, the most secure one is selected. For empty intersections, a new node v is created that shares a nonempty intersection of supported security parameters with nodes $n1$ and $n2$. The original edge $(n1, n2)$ is removed from E and edges $(n1, v)$ and $(v, n2)$ are added, connecting the newly created node. The algorithm then processes the newly added edges. After all edges are processed, the optimal supported security parameters are determined for every node.

Algorithm 4.1 Determine security parameters for a given directed acyclic graph. Optimizing for maximum security.

```
function AnnotateGraph($G = (V, E)$)
    $TransmissionSecParams \leftarrow \varnothing, StorageSecParams \leftarrow \varnothing$
    for all $(n1, n2) \in E$ do
        $sp1 \leftarrow$ ViableSecParams(Capabilities($n1$))
        $sp2 \leftarrow$ ViableSecParams(Capabilities($n2$))
        $i \leftarrow sp1 \cap sp2$            // the security parameters $n1$ and $n2$ have in common
        if $i \neq \varnothing$ then
            $i \leftarrow$ MostSecure($i$)
            $TransmissionSecParams \leftarrow TransmissionSecParams \cup \{(n1, n2, i)\}$
        else
            $v \leftarrow$ NewComponentWith(MostSecure($sp1$), MostSecure($sp2$))
            $E \leftarrow \{(n1, v), (v, n2)\} \cup E \setminus (n1, n2)$
            $V \leftarrow V \cup \{v\}$
            $(\_, TSP', \_) \leftarrow$ AnnotateGraph($(\varnothing, \{(n1, v), (v, n2)\})$)
            $TransmissionSecParams \leftarrow TransmissionSecParams \cup TSP'$
        end if
    end for
    for all $n \in V$ do
        $sp \leftarrow$ MostSecure(ViableSecParams(Capabilities($n$)))
        $StorageSecParams \leftarrow StorageSecParams \cup \{(n, sp)\}$
    end for
     return $(G, TransmissionSecParams, StorageSecParams)$
end function
```

Further modifications to the graph can be made to extend access management. Additional processing steps can be added in case specific *obligations* were imposed by the authorization control. Obligations can mandate certain parts of the data to be modified, which can be useful in various situations. For example, a user is working with a medical data set but should not be able to see names of patients. A processing step can be added to anonymize these at an appropriate time in the data flow, before the user will see any data.

Based on the finalized graph and the order of processing steps, necessary artifacts are generated that are needed during execution. These include unique authentication credentials for every component and user involved but can also include cryptographic keys to allow for secure communication with the security provider.

The attributed graph and generated artifacts together form the *plan*, which is digitally signed to prevent unauthorized modifications. The finished plan is sent to the execution engine in charge of orchestrating the data process and a copy is sent to the *security provider* to handle authentication and authorization requests during execution.

4.4.2 Security Provider

The security provider is a special component forming an integral part of this security concept. It is an instance controlled by the user interested in secure data processing, therefore, considered trusted and used for authentication, authorization, and key management. Despite its numerous tasks, it is specifically designed to have an overall low workload and scales linearly with the number of components and transmissions involved. Note that a *component* abstracts a whole processing step in its entirety, so there are no scaling issues with a highly distributed processing step, as it is considered as one single component.

The security provider maintains a registry of every known user and their respective access privileges, defined in advance. Access privileges for components and users to parts of data during execution follow implicitly from the plan the security provider received. Each component and user is only aware of their direct adjacent neighbors in the graph. Moreover, they are only entitled to receive data transmissions from such nodes they receive a positive inflow from (that is, have an incoming edge). The plan is also used by the security provider to extract authentication credentials and other artifacts that were generated for the components. Further artifacts, such as encryption keys, are then generated for every data transmission, based on the security parameters found in the plan.

During execution, users and components will authenticate with the security provider and receive the necessary artifacts to decrypt incoming data, encrypt outgoing data and handle their stored data, operating based on the plan. This serves as access control per user and component, as they only receive keys and data necessary for their step in the chain of data processing. As the plan is digitally signed, the security provider can validate it if needed because it knows all users.

4.5 Applying the Concept

Following is described how this concept is applied to secure distributed data processing in two phases.

Phase 1: Modeling

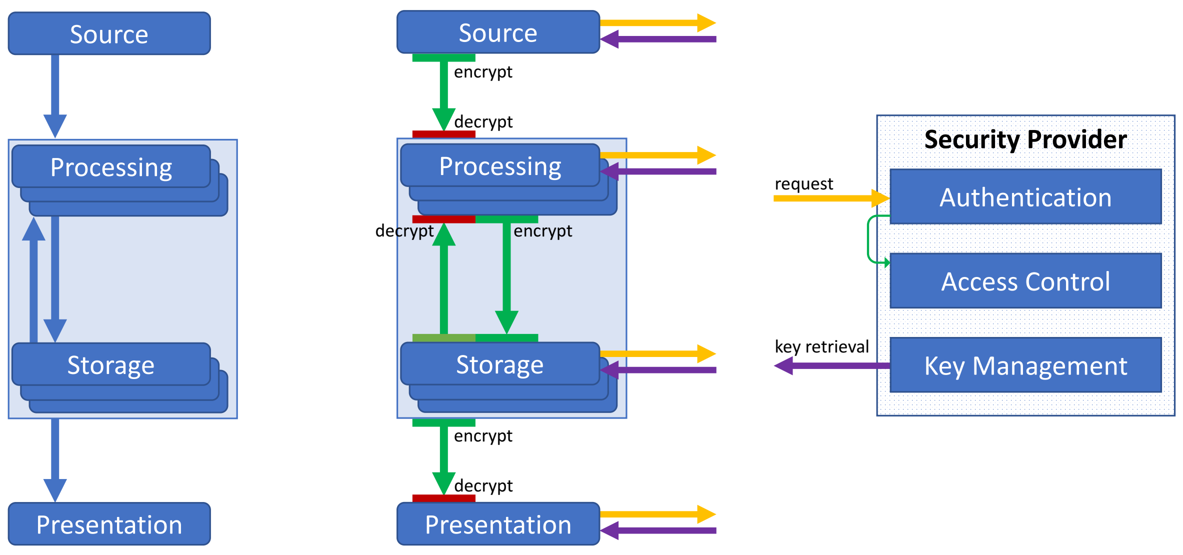
During the modeling phase, a graph suitable for generating a plan (as described in Section 4.4.1) needs to be created. To yield such an input graph, a user can utilize an unspecified tool to either create the whole data process from scratch, or model an input graph for an already defined data process. Such a modeling tool can utilize the security provider for access control and restrict the data sources or data processing steps a specific user is able to use. This input is then transformed into a plan as described above, signed by the modeler and handed over to an execution engine for processing. It is also sent to the security provider. The procedure can be seen in Figure 5.2b.

Note that neither the input graph, defining the data flow with its components, nor the finalized plan, have a fixed format. Both are merely a collection of information and their realization is left to concrete implementations for generating such a plan. One possibility is to utilize already existing formats, such as input formats used by workflow and execution engines. Workflow or execution engines typically operate based on a file format that models processes and their relationships. This model references implementations in a repository of the engine. The engine executes the defined model by running appropriate implementations and routing data flows between defined components, as specified by the given model. Consequently, the output format of a plan, as defined above, can target such an execution engine and realize the determined security parameters by using suitable implementations in the engine's repository. This allows secure distributed data processing with standard execution tools that need not be customized.

Phase 2: Execution

The data flow, as defined in the previous phase, is then processed.This phase is implementation specific, therefore, the following will focus on certain generic key elements. For ease of discussion, it is assumed that processing is done by an executing engine as mentioned in the previous phase.

In order to explain the events taking place during execution, we assume a generalized and simplified abstraction of a data flow. Every executed data flow can be broken down



(a) Insecure execution

(b) Secure execution

Figure 4.3: Insecure (left) and secured (right) data processing side by side. Every component authenticates and, when authentication succeeds, receives information and cryptographic keys.

into a set of abstract flows consisting of four components: data source, presentation of a result, and data processing as well as storage. This is demonstrated in Figure 4.3, in which a classic execution (left) and a secured execution (right) can be seen, following a previously generated plan. It can be seen that all transmissions of data are now encrypted (green). Furthermore, data is decrypted (red) and newly encrypted (green) for each transmission. Appropriately chosen protocols ensure data integrity and authentication of parties involved in the transmission. Data is stored encrypted on dedicated storage services and is also encrypted in temporary storage on processing services. Unless cryptosystems with homomorphic properties or similar algorithms are used (cf. Section 2.2.2), data has to be in plaintext during processing.

At time of provisioning, each component, such as cloud services or other processing facilities, receives its individual, unique authentication information that was created as part of the generated plan. With that, each component can authenticate with the security provider, who is also aware of these credentials because it received a copy of the plan. Before a component attempts to receive data, it contacts the security provider and authenticates. Upon success, the security provider will supply the component with necessary information and credentials to receive and decrypt an incoming data transfer. Because the security provider is aware of the plan and each component is using unique authentication credentials, the legibility of the component's request can be reliably

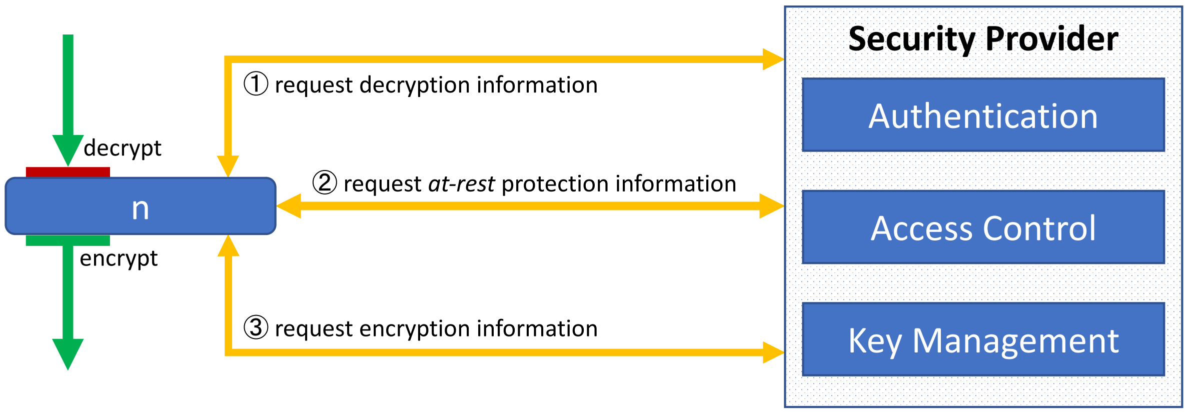

Security Provider
Authentication
Access Control
Key Management
① request decryption information
② request *at-rest* protection information
③ request encryption information
decrypt
n
encrypt

Figure 4.4: Requests made by component n to the security provider.

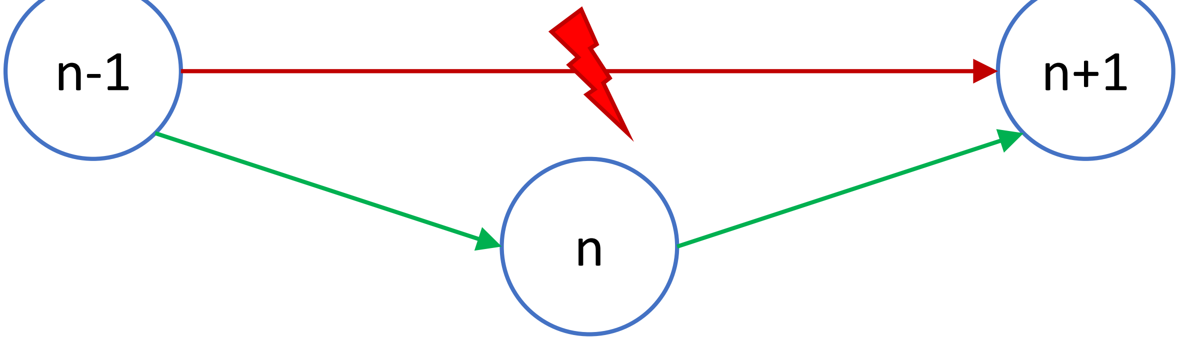


Figure 4.5: n+1 is unable to intercept data from n-1 because it does not posses necessary decryption information.

determined. Further requests are made to the security provider for credentials to protect temporary or permanently stored data. Before data is sent to the next component, one more set of information and credentials is requested, so that the current component knows how to communicate with the target and how to protect the data during transfer. This is illustrated in Figure 4.4.

This makes it impossible for a component to properly receive data transfers intended for other components and implicitly enforces the correct order of the data flow. It is also not possible for a component to produce data transfers that can be used by components other than the ones intended to. This is illustrated in Figure 4.5. Note how component $n-1$ can send valid data to n and n to $n+1$. However, $n+1$ is unable to intercept data from $n+1$. Both components are not aware of each other, the protocol to be used was never specified for this pair of components and, therefore, would have to be guessed. In case certain restrictions can be made to the determination of security parameters, such as using asymmetric cryptosystems, reverse data transfers from $n+1$ to n and n to $n-1$ can also be reliably prevented.

Distributed Processing

A component is an abstraction of a processing step and as such it can also represent a distributed processing step, for example, an instance of Hadoop. Even though it consists of potentially many compute nodes internally, such a case does not need special handling. Receiving and transmitting protected data from and to other components works as described above. For component-internal security, it is assumed the used distributed processing framework is equipped with appropriate functionality to protect data at-rest in-between processing steps. This is the case for Hadoop: authentication is supported using Kerberos [DORZ11], encryption for data at-rest is available in HDFS [Apa17] and extended versions with stronger security guarantees and access control are available as well, for example, G-Hadoop [ZWT+14].

Reducing Flexibility

Even though this security concept focuses on flexibility regarding the applied security parameters, it is worth mentioning that optionally limiting the cryptographic ciphers to proxy re-encryption schemes (cf. Section 2.2.1) has advantages.

The major advantage is the prevention of one plaintext step. Considering a component n with encrypted data in either temporary or permanent storage: this data needs to be decrypted and then encrypted again for the next component $n+1$, before it can be transmitted. Between decryption and encryption, the data is in an unencrypted state for a short time, potentially only in memory. In case a proxy re-encryption scheme is used, the component would request a re-encryption key $RK_{n \to n+1}$ from the security provider, instead of a decryption and new encryption key. By using re-encryption instead of decryption and encryption, the data will never be unencrypted between storage and transfer to the next component $n+1$. Additionally, depending on the actual cryptosystem being used, re-encrypting is usually a less computationally expensive operation.

4.6 Recommendations

As a user of cloud services, one is subject to the features and capabilities offered by the provider. This is especially true for SaaS offerings but also for IaaS, where customers are often limited to configurations with respect to their VMs, while the surrounding architecture is out of control. This is necessary, of course, to provide proper user isolation but it also limits customers in what they can do. It also creates uncertainties regarding the promises made by a provider. Even though the presented security concept is designed to be used with arbitrary cloud services, given the choice, a customer should select offerings that help to increase security overall and reduce risks.

Open standards in authentication and authorization, such as the OASIS's Security Assertion Markup Language (SAML)[1] and XACML[2], are more likely to be widely supported and receive good implementations, compared to self-made solutions. As mentioned in Section 3.2, web-facing technologies have to be properly hardened [Jan11] and secured by the provider [AGM16]. This also implies classic security technologies, such as web-application scanners, penetration testing and firewalls.

Because virtualization security is usually completely out of control for customers, it is even more important to pay attention to the virtualization technology a provider uses. Solutions to secure these parts are presented in Section 3.2 and customers can prefer cloud providers implementing such systems. Additionally, the Cloud Security Alliance [Clo17] published recommendations regarding virtualization security that can help customers assessing the risks they are exposed to with a certain provider.

One more important aspect is audit and monitoring. Users should be aware of the auditing capabilities a provider makes available to its customers, regarding their data and the services they use. Several possibilities to use such capabilities, in order to increase security, are described in Section 3.2.

[1]XML-based format for exchanging authentication information [Org17c]
[2]Language, architecture and processing model for fine-grained access management [Org17a]

5 Implementation in FlexMash

The presented security concept is designed to be generic and applicable to many distributed data processing systems. FlexMash, as a tool for data mashups (cf. Section 2.1), is no exception. This chapter describes how the security concept can be implemented in FlexMash, down to a technical level. Extending plan generation is described, along with an exemplary setup for a security provider (cf. Section 4.4.2). Finally, it is explained how FlexMash operates with the security concept.

5.1 Overview

Figure 5.1 shows FlexMash's steps. User interaction takes place during steps one and two. Consequently, these steps require a form of access management. Steps three and four are algorithmic decisions and processing of the mashup plan itself. Actual data processing — that needs to be protected — happens in the course of steps five and six.

5.2 Plan

Because FlexMash already contains a modeling and transformation step, it can be extended to also determine the security parameters and artifacts needed for the security

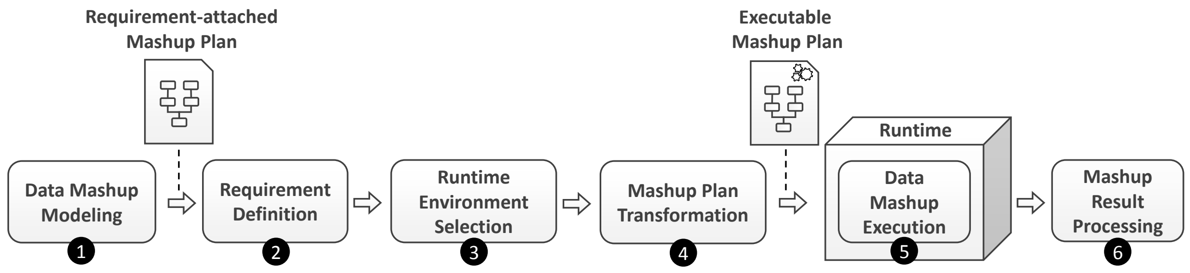


Figure 5.1: The six steps of a mashup in FlexMash, from modeling to result processing. Illustration by Hirmer and Behringer [HB17].

concept (cf. Section 4.4.1). After a user modeled a mashup, non-functional requirements are selected. *Security* can be added as one such requirement. When a user selects it, the security concept is applied to the modeled mashup. Combining various requirements such as security and prioritized execution speed can influence the selection of determined security parameters, for example, by preferring hardware accelerated cryptosystems. As mentioned in Section 4.4.1, this can be accomplished by different weighting of parameters in the function `MostSecure` of Algorithm 4.1.

The presented Algorithm 4.1 is added to FlexMash's transformation step, in order to modify the modeled workflow where needed and add appropriate security information to the executable plan. At this point, the non-functional requirement selection already took place and it is known if the mashup has to be secured. For the algorithm to work, capabilities of servers need to be supplied as input. This can be accomplished by maintaining a list or database with the capabilities of all cloud providers available to this user. Alternatively, OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) can be utilized. Its usage is proposed by Hirmer and Behringer [HB17] to automatically deploy cloud components on-demand, as needed by a mashup. TOSCA supports modeling cloud components with `ServerProperties` and, therefore, already has the necessary information at its disposal, rendering an external list of capabilities redundant [Org17d]. In FlexMash's mashup plan transformation step, secure DSD and DPD implementations are selected that support the determined security parameters, such as encryption algorithms, digital signature schemes, and protocols.

The plan's transformation to an executable format can target, among others, WS-BPEL or BPMN, as mentioned in Section 2.1. Generated security artifacts can be added as variable or payload for the respective process. This allows to supply a process with the necessary credentials needed to authenticate with the security provider during execution, or cryptographic keys to communicate securely with it. For example, a unique RSA key pair can be created for each process and the respective process is supplied the private key for authentication, secure communication with the security provider, and digitally signing data transfers.

5.3 Security Provider

The security provider is a lightweight server that provides authentication, authorization and key management services. Along the recommendations made in Section 4.6, solutions based on open standards have considerable advantages and are, therefore, used for this suggested implementation.

For authentication, OASIS SAML [Org17c] is used as an open standard for exchanging authentication information. This defined data format is based on XML and is supported by a broad range of available software and services, including cloud providers, such as Amazon [Ama17]. SAML defines *assertions*, *protocols*, *bindings* and *profiles*. Assertions are used for authentication statements. They provide information about the authentication status and can communicate additional attributes about a user. Assertions can also be used to communicate authorization statements. However, these are not comparable to the expressive power of XACML. For more complex authorization scenarios, SAML explicitly supports interoperability with XACML. Protocols define how SAML entities have to handle SAML elements, such as that a query requires a corresponding response. Bindings specify the communication protocol used to send a SAML message, such as SOAP. Profiles are a set of predefined assertions, protocols, and bindings.

While SAML is mostly known for SSO, this is not the only mode of operation it supports. The generic design of SAML knows three parties: a user, a service provider and an identity provider where both — service provider and identity provider — have to support SAML. Upon requesting a resource from a service provider, the user will be redirected to an identity provider for authentication. The result of this authentication is then passed back to the service provider. In light of this security concept, the security provider is both service provider and identity provider, while components are users. The resources provided to the components consist of the necessary means and information to access and protect data.

OASIS XACML [Org17a] handles authorization for users that were authenticated by SAML. Upon authorization requests, the request is evaluated against pre-configured access policies. The reached decision is then returned.

The internal architecture of XACML separates these tasks among five so called *points*. A request is sent to the Policy Enforcement Point (PEP), converted to an internal format and sent to the Policy Decision Point (PDP) for evaluation. The PDP evaluates against access policies retrieved from the Policy Retrieval Point (PRP) and Policy Information Point (PIP), managed by the Policy Administration Point (PAP). The evaluated result is returned to the PEP.

XACML supports obligations. These are instructions that need to be followed in order for access to be granted. For example, a user can get access to sensitive data but only when logging is turned on and it can be tracked what this user is reading. The paradigm of access control supported by XACML is attribute-based access control. It works by associating users and resources with attributes and allows to express complex access policies.

OASIS KMIP [Org17b] is used for key management. Certificates, cryptographic keys, and custom data are stored on the security provider as *managed objects* and — given a successful authentication and authorization — communicated to the requesting component or user. KMIP supports a wide range of operations, such as creation, retrieval, registration, replacement, certification, decryption, encryption, export, and import of its managed objects. Keys can be wrapped in order to transmit them encrypted.

5.4 Applying the Concept

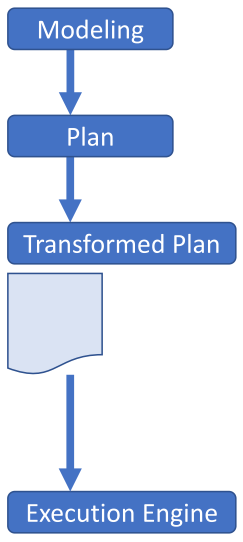
Following, FlexMash operating with the applied security concept is described, broken down into the two phases of the concept.

Phase 1: Modeling

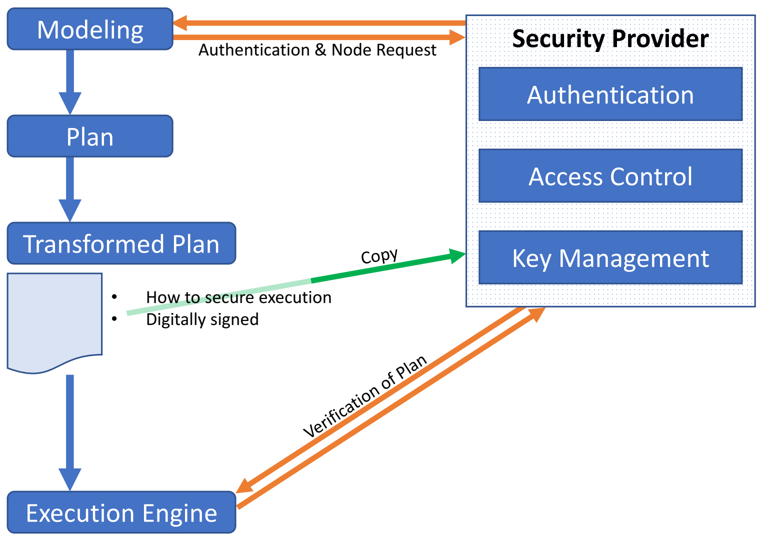
The modeling step of FlexMash is extended by adding an authentication to the used modeling tool. The security provider has records for all users and their access privileges with regard to DSDs and DPDs. Furthermore, the DSDs and DPDs themselves are stored by the security provider. Upon successful authentication, the modeling tool is supplied with the definitions of DSDs and DPDs this user is allowed to use. After modeling is completed, non-functional requirements are selected. For the sake of this description, it is assumed that a secure execution is requested.

The modeled mashup plan is now transformed into an executable plan according to descriptions in Section 5.2. After this plan is generated, it is digitally signed by the modeling user and passed to the selected engine for execution. Additionally, a copy is transferred to the security provider. Figure 5.2 illustrates this process. The insecure plan modeling (left) is extended by an authentication procedure (right). Before execution takes place, the engine can verify the integrity and consistency of the plan with the security provider, if supported.

As soon as the security provider receives the plan, it starts analyzing it: an identity is created for each component so they can authenticate with the security provider. Previously generated artifacts, such as authentication credentials, are extracted from the plan and further artifacts, such as encryption keys, are generated for each component. In addition to that, the security provider can validate the consistency of the plan and make sure that all components used are within the access privileges of the modeling user. If this is not the case, the security provider can refuse to validate the plan for the execution engine. It can also refuse to provide requested information and artifacts to the components, effectively preventing execution.



(a) Previous plan generation



(b) Plan generation with security concept

Figure 5.2: Prior plan generation (left) and new plan generation (right) in FlexMash adopting the security concept.

It is important to highlight that until now, no data was processed or transferred. The modeled mashup plan contains only DSDs and DPDs that are covered by the modeler's access privileges. Consequently, the execution is ensured to comply with access policies as it cannot contain any DSDs and DPDs other than the ones the respective modeler can access. This provides access management, enforced on the modeling level. The advantage of this approach is that data does not need to be protected with complex techniques in order to make selective access from different users possible. Rather, a user with insufficient access privileges is unable to model a mashup with this data, unable to commence its usage in the first place. More fine-grained permissions can be accomplished by providing multiple DSDs that are different views of the same data source.

Even more flexibility for access control can be achieved by utilizing *subflows*, a feature Hirmer and Behringer [HB17] added to FlexMash. It allows to use abstract nodes that represent a whole mashup plan. When combining this feature with XACML's obligations, the security provider can return a node that imposes restrictions on the usage of the requested resource. For example, when requesting a data source with sensitive medical data, the security provider can allow a user to use this source but only when various data fields were anonymized. For this purpose, a subflow node is returned that contains the requested data source, along with a DPD, to filter sensitive information before the data can be used for anything else. This is illustrated in Figure 5.3.

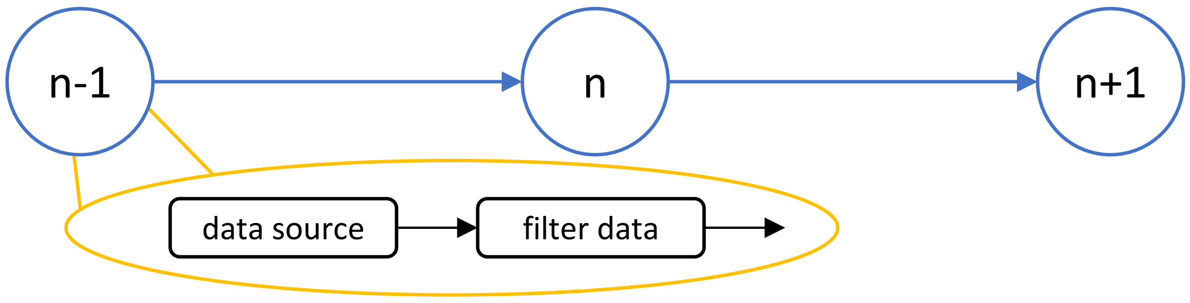


Figure 5.3: Node `n-1` is a subflow and consists of multiple nodes internally, in order to realize an imposed obligation.

Phase 2: Execution

In the final phase, the previously selected engine executes the generated plan. Following the provided plan, components are provisioned as needed and supplied with authentication credentials contained in the plan. The components contact the security provider, authenticate and request artifacts and information about how to handle incoming data, outgoing data and data at-rest, as illustrated in Figure 4.3. This implicitly enforces that a component can only access data it is intended to process, based on the execution order of the plan.

Due to the approach of this security concept, standard formats can still be used for the executable mashup plan. The only requirement is for the data format to support input variables for processes in some way. This is not unreasonable, considering formats like WS-BPEL are expressive on purpose. As a result, workflow engines supporting the selected executable format can be used without modifications.

6 Evaluation

Based on literature research, requirements were defined in Section 4.1. In order to evaluate the proposed security concept, the fulfillment of these requirements is examined. Because certain aspects are implementation specific, the evaluation is based on the suggested implementation in FlexMash (cf. Chapter 5).

Req_1 Access control for users

The security provider, as a central trusted entity, manages available components that are used for modeling a data flow, as explained in Section 5.4. Only authenticated users can access components and the security provider will only grant requests for known parties with sufficient access privileges.

Modeling a data flow is done using a tool that communicates with the security provider and runs within the trusted area of a user. Consequently, when the security provider receives the digitally signed plan with security information, it can be assumed to be genuine. However, even a forged plan from a malicious modeling tool cannot lead to violations of security. The security provider has all data needed to validate the plan and make sure the modeling user's access privileges are not exceeded throughout the data flow. In case such a violation is found, the security provider can either inform the execution engine by not validating the plan, or refuse responses to requests from components during execution, withholding information needed to handle data and, therefore, effectively preventing execution of the data flow.

Req_2 Access control for processing components

A given component is either a data source, a data sink (consumer), or processes data. These components are short-lived in that they are provisioned on-demand and carry out their processing for one single user, the modeler of a data flow. They are not long-lived environments with multiple users that share data among each other. Consequently, as only one user exists and each component only fulfills one purpose, access management within a component is not necessary.

This leaves data transmissions for consideration. It has to be ensured that each component can only access incoming data it is entitled to. Data sources do not have incoming transmissions and only operate based on data they already know. Data sinks and processing components, however, do receive data and it must be ensured they only receive such data they are entitled to handle. As per the defined security plan (cf. Section 4.4.1), all incoming data transmissions are encrypted. The security provider also makes sure that each transmission is encrypted with a unique set of cryptographic keys (cf. Section 4.4.2). For a data sink or processing component to be able to decrypt an incoming data transmission, the security provider has to be contacted, as it is the only entity with the necessary information. However, the security provider only supplies a component with information and cryptographic keys, when the component can successfully authenticate and the data transmission in question is consistent with the modeled data flow. Consequently, a component that can either not authenticate successfully or is not supposed to receive this data transmission, is not supplied with the information necessary to handle an incoming data transfer.

Req_3 Confidentiality

The confidentiality of data can be evaluated in three states: in-transit, at-rest and in-use. The proposed security concept protects the confidentiality of each state in the same way: by specifying the security measures that have to be taken.

Due to all transmissions being cryptographically secured, confidentiality is maintained in-transit. Transmissions between components are encrypted as defined by the plan and cryptographic artifacts retrieved from the security provider are encrypted as well, so they cannot be intercepted and stolen. Protection for data at-rest, such as encryption, is specified in the plan as well. Because resting data is encrypted, snapshots and backups of data, unwittingly performed by a cloud provider, are of no concern. Once cryptographic keys are destroyed after data was processed, leftover data in the cloud can be considered inaccessible, given the used cryptographic cipher is strong enough.

Theoretically, confidentiality for data in-use can be provided as well. However, at the time of writing, cryptosystems providing necessary features, such as homomorphic properties (cf. Section 2.2.2), are not yet suitable for data processing. Once they are, data in-use can be protected by the existing mechanisms of this security concept, analogous to data in-use and at-rest.

The assumption of encryption being carried out is reliable because the implementations of components are code supplied by the user or — in the case of FlexMash — provided by a trusted technical expert. Consequently, collusion between deployed components can be ruled out as well.

Req_4 Integrity

Analogous to the use of encryption to ensure confidentiality for data in all considered states (in-transit, at-rest), digital signatures are used to ensure integrity of data. Each component and user has a unique identity, tied to one data flow, that can be used to sign transmitted and stored data. This allows to reliably detect modifications from adversaries.

Req_5 Identification

The generation of the plan ensures that for every component and user taking part in the data flow, unique authentication credentials are produced. This makes it possible to uniquely identify them even among components part of past and future data flows.

Req_6 Heterogeneous environment

The proposed security concept does not require specific features or capabilities from a cloud service other than means of secure communication. This is trivial for IaaS solutions as implementations can be provided by the customer. In case of PaaS, the possibilities are more constrained. However, this is not an issue as the security concept does not require one specific security feature but rather works with what is available. The situation is more complex with SaaS solutions. Although even in this case, basic means of secure communication, such as SSL/TLS, are usually available. The algorithmic generation of a secure plan, connecting different cloud solutions based on their provided capabilities, makes sure to allow secure interoperability of heterogeneous cloud solutions.

Req_7 Adjustable security requirements

The employed means of security depend on the selection of security parameters that are available to a component. In the provided implementation for graph annotation (cf. Algorithm 4.1), this selection is performed by the function `MostSecure`, which optimizes for security. Swapping this function with an alternative can optimize for different goals.

Req_8 Flexibility

The proposed security concept operates independent of specific cryptosystems or algorithms. Determining the security parameters during plan generation works by choosing an *optimal* algorithm from an arbitrary set of available algorithms, with the definition of optimal being up to the user.

Req_9 Privacy

In case sensitive data is involved and privacy has to be ensured, the security provider can issue obligations along permissions for a certain resource. These obligations will modify data to ensure the desired level of privacy. As obligations consist of components themselves, they are able to perform arbitrarily complex data operations.

Req_{10} Scalability and distributed processing

The bottleneck of this concept is the introduced entity called security provider. It is a required part for the security of this concept. Event though it is theoretically feasible to work without a dedicated security provider by directly supplying each component with all necessary artifacts upon provisioning, this inevitably decreases security.

The tasks performed by the security provider are not exceedingly computationally expensive, with the possible exception of artifact and key generation after a plan was submitted to it. However, this only happens once for each plan. During execution, the workload is reduced to authentication and communication duties. This communication is limited by the number of components and data transfers involved and, therefore, scales linearly with the size of a data flow. Note that for sequential parts in a data flow, the security provider does not have to serve concurrent requests. Moreover, components are an abstraction of their underlying processing entity and a highly distributed processing step is only considered as one single component. Overall, the expected workload on the security provider can be considered minimal and does not impede the scalability of distributed data processing.

7 Conclusion and Future Work

The cloud is a field incorporating a large and diverse set of technologies. And while this leads to the many benefits cloud services provide, it also brings together the security concerns of each of these technologies. For users who want to use the cloud for data processing and entrust their data to the cloud, this is a major hindrance in adopting cloud solutions.

To asses this problem, literature research was performed and many threats and vulnerabilities could be identified that threaten a customer of cloud services. From data leakage in VMs to insecure authentication of web services, all parts of a modern cloud architecture can be attacked. Fortunately, solutions and strategies exist to counter various problems. However, as plentiful as existing security issues are, as many solutions exist, usually as countermeasure against only a small set of threats. Only few holistic solutions to security in the cloud exist, often imposing various limitations that restrict their field of application.

This work proposes a security concept that can be employed by a customer of cloud solutions. It attempts to provide security for all parts of distributed data processing in the cloud. To simultaneously support a heterogeneous and large set of existing cloud solutions, without making many assumptions about the used services, was one of the objectives. In order to reach this goal, the proposed concept performs a static analysis of the data flow to be processed and then dynamically determines the best way to secure this flow. Additionally, recommendations for selection and usage of cloud providers are given, as some security concerns cannot be addressed by a customer, due to the limited control over hardware and software. The concept is designed to be generic in order to allow easy implementation into existing data processing tools. As proof of example, a possible implementation for the mashup tool FlexMash is provided. The security concept is evaluated based on requirements that were derived from literature research on security issues related to cloud technologies. Requirements, such as fine-grained access management, data confidentiality and integrity, flexibility, and scalability were found to be met by the proposed concept.

For future work, a way to uniquely identify a VM can increase security. As a VM can be easily copied by a cloud provider, authenticating credentials contained in this VM are copied as well. Such a copy results in a clone indistinguishable from the original. Values, such as a MAC address, are ineffective for VMs.

As the security concept already performs an analysis in order to determine optimal security parameters, future work can extend this step to also perform optimizations on the data flow. This can help mitigate the performance penalty due to encryption. One more challenge for future work is a way to qualify the security achieved with a specific combination of used algorithms. This would make optimizing for maximum security easier and potentially provable.

Furthermore, in regard to the suggested implementation in FlexMash, a visual cue for the modeler, indicating the security of a mashup, can be a useful extension.

Bibliography

[ABG+05] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, Y. Xu. "Two can keep a secret: A distributed architecture for secure database services." In: *CIDR 2005* (2005) (cit. on p. 28).

[AGM16] M. Almorsy, J. C. Grundy, I. Müller. "An Analysis of the Cloud Computing Security Problem." In: *CoRR* abs/1609.01107 (2016). URL: http://arxiv.org/abs/1609.01107 (cit. on pp. 21, 31, 50).

[Ama17] Amazon. *Enabling SAML 2.0 Federated Users to Access the AWS Management Console*. July 31, 2017. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_enable-console-saml.html (cit. on p. 53).

[Apa17] Apache Software Foundation. *Transparent Encryption in HDFS*. July 31, 2017. URL: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/TransparentEncryption.html (cit. on pp. 36, 49).

[APST12] M. A. AlZain, E. Pardede, B. Soh, J. A. Thom. "Cloud Computing Security: From Single to Multi-clouds." In: *45th Hawaii International Conference on System Sciences*. Institute of Electrical and Electronics Engineers (IEEE), Jan. 2012. DOI: 10.1109/hicss.2012.153 (cit. on p. 25).

[BBS98] M. Blaze, G. Bleumer, M. Strauss. "Divertible Protocols and Atomic Proxy Cryptography." In: *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*. 1998, pp. 127–144. DOI: 10.1007/BFb0054122. URL: https://doi.org/10.1007/BFb0054122 (cit. on p. 17).

[BCQ+13] A. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa. "DepSky: Dependable and Secure Storage in a Cloud-of-Clouds." In: *Trans. Storage* 9.4 (Nov. 2013), 12:1–12:33. ISSN: 1553-3077. DOI: 10.1145/2535929. URL: http://doi.acm.org/10.1145/2535929 (cit. on p. 35).

[BEP+14] J. Bacon, D. Eyers, T. F. J.-M. Pasquier, J. Singh, I. Papagiannis, P. Pietzuch. "Information Flow Control for Secure Cloud Computing." In: *IEEE Transactions on Network and Service Management* 11.1 (Mar. 2014), pp. 76–89. DOI: 10.1109/tnsm.2013.122313.130423 (cit. on p. 35).

[BGV12] Z. Brakerski, C. Gentry, V. Vaikuntanathan. “(Leveled) Fully Homomorphic Encryption Without Bootstrapping.” In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS ’12. Cambridge, Massachusetts: ACM, 2012, pp. 309–325. ISBN: 978-1-4503-1115-1. DOI: 10.1145/2090236.2090262. URL: http://doi.acm.org/10.1145/2090236.2090262 (cit. on p. 18).

[BSR11] A. Bisong, Syed, M. Rahman. “An Overview of the Security Concerns in Enterprise Cloud Computing.” In: *International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.1, January 2011* (Jan. 28, 2011). DOI: 10.5121/ijnsa.2011.3103. arXiv: 1101.5613v1 [cs.CR] (cit. on pp. 21, 23).

[BSW07] J. Bethencourt, A. Sahai, B. Waters. “Ciphertext-Policy Attribute-Based Encryption.” In: *Proceedings of the 2007 IEEE Symposium on Security and Privacy*. SP ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 321–334. ISBN: 0-7695-2848-1. DOI: 10.1109/SP.2007.11. URL: http://dx.doi.org/10.1109/SP.2007.11 (cit. on p. 27).

[CCH+12] S. S. M. Chow, C.-K. Chu, X. Huang, J. Zhou, R. H. Deng. “Cryptography and Security.” In: ed. by D. Naccache. Berlin, Heidelberg: Springer-Verlag, 2012. Chap. Dynamic Secure Cloud Storage with Provenance, pp. 442–464. ISBN: 978-3-642-28367-3. URL: http://dl.acm.org/citation.cfm?id=2184081.2184115 (cit. on p. 30).

[CLH14] P. Chung, C. Liu, M. Hwang. “A Study of Attribute-based Proxy Re-encryption Scheme in Cloud Environments.” In: *I. J. Network Security* 16.1 (2014), pp. 1–13. URL: http://ijns.femto.com.tw/contents/ijns-v16-n1/ijns-2014-v16-n1-p1-13.pdf (cit. on p. 17).

[Clo17] Cloud Security Alliance. *Best Practices for Mitigating Risks in Virtualized Environments*. July 31, 2017. URL: https://downloads.cloudsecurityalliance.org/whitepapers/Best_Practices_for%20_Mitigating_Risks_Virtual_Environments_April2015_4-1-15_GLM5.pdf (cit. on p. 50).

[CM10] R. Chandramouli, P. Mell. “State of Security Readiness.” In: *Crossroads* 16.3 (Mar. 2010), pp. 23–25. ISSN: 1528-4972. DOI: 10.1145/1734160.1734168. URL: http://doi.acm.org/10.1145/1734160.1734168 (cit. on p. 21).

[DJNP13] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter. “Secure Key Management in the Cloud.” In: *Proceedings of the 14th IMA International Conference on Cryptography and Coding - Volume 8308*. IMACC 2013. Oxford, UK: Springer-Verlag New York, Inc., 2013, pp. 270–289. ISBN: 978-3-642-45238-3. DOI: 10.1007/978-3-642-45239-0_16. URL: http://dx.doi.org/10.1007/978-3-642-45239-0_16 (cit. on p. 28).

[DORZ11] D. Das, O. O'Malley, S. Radia, K. Zhang. "Adding security to apache hadoop." In: *Hortonworks, IBM* (2011) (cit. on pp. 26, 36, 49).

[DTM10] W. Dawoud, I. Takouna, C. Meinel. "Infrastructure as a service security: Challenges and solutions." In: *2010 The 7th International Conference on Informatics and Systems (INFOS)*. Mar. 2010, pp. 1–8 (cit. on pp. 21, 23, 25, 31).

[El 85] T. El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." In: (1985), pp. 10–18. URL: http://dl.acm.org/citation.cfm?id=19478.19480 (cit. on p. 18).

[Ele17] Electronic Frontier Foundation. *Trusted Computing Promise and Risk*. July 31, 2017. URL: https://www.eff.org/wp/trusted-computing-promise-and-risk (cit. on p. 38).

[GR05] T. Garfinkel, M. Rosenblum. "When Virtual is Harder Than Real: Security Challenges in Virtual Machine Based Computing Environments." In: *Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10*. HOTOS'05. Santa Fe, NM: USENIX Association, 2005, pp. 20–20. URL: http://dl.acm.org/citation.cfm?id=1251123.1251143 (cit. on pp. 23, 24, 33).

[GR12] J. Gantz, D. Reinsel. "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east." In: *IDC iView: IDC Analyze the future* 2007.2012 (2012), pp. 1–16 (cit. on p. 13).

[GWS11] B. Grobauer, T. Walloschek, E. Stocker. "Understanding Cloud Computing Vulnerabilities." In: *IEEE Security and Privacy* 9.2 (Mar. 2011), pp. 50–57. ISSN: 1540-7993. DOI: 10.1109/MSP.2010.115. URL: http://dx.doi.org/10.1109/MSP.2010.115 (cit. on pp. 19, 20, 22, 24).

[Hal17] S. Halevi. *HElib*. July 31, 2017. URL: https://shaih.github.io/HElib/ (visited on 04/01/2017) (cit. on p. 18).

[HB17] P. Hirmer, M. Behringer. "FlexMash 2.0 – Flexible Modeling and Execution of Data Mashups." In: *Communications in Computer and Information Science*. algorithms. Springer Nature, 2017, pp. 10–29. DOI: 10.1007/978-3-319-53174-8_2 (cit. on pp. 15, 16, 51, 52, 55).

[HKKT10] K. Hamlen, M. Kantarcioglu, L. Khan, B. Thuraisingham. "Security Issues for Cloud Computing." In: *International Journal of Information Security and Privacy* 4.2 (2010), pp. 36–48. DOI: 10.4018/jisp.2010040103 (cit. on pp. 37, 38, 41).

[HM16] P. Hirmer, B. Mitschang. "Flexible Data Mashups Based on Pattern-Based Model Transformation." In: *Communications in Computer and Information Science*. Springer Nature, 2016, pp. 12–30. DOI: 10.1007/978-3-319-28727-0_2 (cit. on pp. 13, 15, 16).

[HRFF13] K. Hashizume, D. G. Rosado, E. Fernández-Medina, E. B. Fernandez. "An analysis of security issues for cloud computing." In: *Journal of Internet Services and Applications* 4.1 (2013), p. 5. DOI: 10.1186/1869-0238-4-5 (cit. on pp. 19, 20, 22, 24, 25).

[Jan11] W. A. Jansen. "Cloud Hooks: Security and Privacy Issues in Cloud Computing." In: *2011 44th Hawaii International Conference on System Sciences*. Jan. 2011, pp. 1–10. DOI: 10.1109/HICSS.2011.103 (cit. on pp. 20, 22, 26, 31, 50).

[KAF+08] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, G. Melançon. "Visual Analytics: Definition, Process, and Challenges." In: *Information Visualization: Human-Centered Issues and Perspectives*. Ed. by A. Kerren, J. T. Stasko, J.-D. Fekete, C. North. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 154–175. ISBN: 978-3-540-70956-5. DOI: 10.1007/978-3-540-70956-5_7. URL: https://doi.org/10.1007/978-3-540-70956-5_7 (cit. on p. 13).

[KL10] S. Kamara, K. Lauter. "Cryptographic Cloud Storage." In: *Financial Cryptography and Data Security*. Springer Nature, 2010, pp. 136–149. DOI: 10.1007/978-3-642-14992-4_13 (cit. on p. 27).

[LAL+14] K. Liang, M. H. Au, J. K. Liu, W. Susilo, D. S. Wong, G. Yang, T. V. X. Phuong, Q. Xie. "A DFA-Based Functional Proxy Re-Encryption Scheme for Secure Public Cloud Data Sharing." In: *IEEE Transactions on Information Forensics and Security* 9.10 (Oct. 2014), pp. 1667–1680. DOI: 10.1109/tifs.2014.2346023 (cit. on p. 29).

[LP11] F. Lombardi, R. D. Pietro. "Secure virtualization for cloud computing." In: *Journal of Network and Computer Applications* 34.4 (July 2011), pp. 1113–1122. DOI: 10.1016/j.jnca.2010.06.008 (cit. on p. 33).

[LSTL12] H.-Y. Lin, S.-T. Shen, W.-G. Tzeng, B.-S. P. Lin. "Toward Data Confidentiality via Integrating Hybrid Encryption Schemes and Hadoop Distributed File System." In: *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*. Institute of Electrical and Electronics Engineers (IEEE), Mar. 2012. DOI: 10.1109/aina.2012.28 (cit. on p. 36).

[LZJ10] S. Lei, D. Zishan, G. Jindi. “Research on Key Management Infrastructure in Cloud Computing Environment.” In: *2010 Ninth International Conference on Grid and Cloud Computing*. Institute of Electrical and Electronics Engineers (IEEE), Nov. 2010. DOI: 10.1109/gcc.2010.84 (cit. on p. 29).

[MKL09] T. Mather, S. Kumaraswamy, S. Latif. *Cloud security and privacy: an enterprise perspective on risks and compliance*. O’Reilly Media, Inc., 2009 (cit. on p. 23).

[NLV11] M. Naehrig, K. Lauter, V. Vaikuntanathan. “Can Homomorphic Encryption Be Practical?” In: *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. CCSW ’11. Chicago, Illinois, USA: ACM, 2011, pp. 113–124. ISBN: 978-1-4503-1004-8. DOI: 10.1145/2046660.2046682. URL: http://doi.acm.org/10.1145/2046660.2046682 (cit. on pp. 18, 34).

[Org17a] Organization for the Advancement of Structured Information Standards. *OASIS eXtensible Access Control Markup Language*. July 31, 2017. URL: https://www.oasis-open.org/committees/xacml (cit. on pp. 50, 53).

[Org17b] Organization for the Advancement of Structured Information Standards. *OASIS Key Management Interoperability Protocol*. July 31, 2017. URL: http://www.oasis-open.org/committees/kmip (cit. on pp. 29, 53).

[Org17c] Organization for the Advancement of Structured Information Standards. *OASIS Security Services*. July 31, 2017. URL: https://www.oasis-open.org/committees/security (cit. on pp. 50, 53).

[Org17d] Organization for the Advancement of Structured Information Standards. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*. July 31, 2017. URL: http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf (cit. on p. 52).

[PBT+10] K. Puolamäki, A. Bertone, R. Therón, O. Huisman, J. Johansson, S. Miksch, P. Papapetrou, S. Rinzivillo. “Data Mining.” In: D. Keim, J. Kohlhammer, G. Ellis, F. Mansmann. *Mastering the Information Age: Solving Problems with Visual Analytics*. 2010, pp. 39–56 (cit. on p. 13).

[RCT+11] A. Rahumed, H. C. Chen, Y. Tang, P. P. Lee, J. C. Lui. “A Secure Cloud Backup System with Assured Deletion and Version Control.” In: *2011 40th International Conference on Parallel Processing Workshops*. Institute of Electrical and Electronics Engineers (IEEE), Sept. 2011. DOI: 10.1109/icppw.2011.17 (cit. on p. 27).

[RLP+14] P. Rad, V. Lindberg, J. Prevost, W. Zhang, M. Jamshidi. “ZeroVM: secure distributed processing for big data analytics.” In: *2014 World Automation Congress (WAC)*. Institute of Electrical and Electronics Engineers (IEEE), Aug. 2014. DOI: 10.1109/wac.2014.7084334 (cit. on p. 33).

[RTSS09] T. Ristenpart, E. Tromer, H. Shacham, S. Savage. “Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds.” In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS ’09. Chicago, Illinois, USA: ACM, 2009, pp. 199–212. ISBN: 978-1-60558-894-0. DOI: 10.1145/1653662.1653687. URL: http://doi.acm.org/10.1145/1653662.1653687 (cit. on p. 25).

[SGR09] N. Santos, K. P. Gummadi, R. Rodrigues. “Towards Trusted Cloud Computing.” In: *HotCloud* 9.9 (2009), p. 3 (cit. on p. 32).

[SKLR11] J. Szefer, E. Keller, R. B. Lee, J. Rexford. “Eliminating the Hypervisor Attack Surface for a More Secure Cloud.” In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS ’11. Chicago, Illinois, USA: ACM, 2011, pp. 401–412. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046754. URL: http://doi.acm.org/10.1145/2046707.2046754 (cit. on p. 34).

[Sta17] R. Stallman. *Can You Trust Your Computer?* July 31, 2017. URL: https://www.gnu.org/philosophy/can-you-trust.html (cit. on p. 38).

[TLLP12] Y. Tang, P. P. Lee, J. C. Lui, R. Perlman. “Secure Overlay Cloud Storage with Access Control and Assured Deletion.” In: *IEEE Transactions on Dependable and Secure Computing* 9.6 (Nov. 2012), pp. 903–916. DOI: 10.1109/tdsc.2012.49 (cit. on p. 30).

[Tru17] Trusted Computing Group. *Trusted Platform Module Library Specification*. July 31, 2017. URL: https://trustedcomputinggroup.org/tpm-library-specification/ (cit. on p. 38).

[VSWW10] K. Vieira, A. Schulter, C. Westphall, C. Westphall. “Intrusion detection techniques in grid and cloud computing environment.” In: *IT Professional, IEEE Computer Society* 12.4 (2010), pp. 38–43 (cit. on p. 32).

[Wai17] T. Waizenegger. “Secure Cryptographic Deletion in the Swift Object Store.” In: (2017) (cit. on p. 27).

[WCW+13] C. Wang, S. S. Chow, Q. Wang, K. Ren, W. Lou. “Privacy-Preserving Public Auditing for Secure Cloud Storage.” In: *IEEE Transactions on Computers* 62.2 (Feb. 2013), pp. 362–375. DOI: 10.1109/tc.2011.245 (cit. on p. 31).

[WDWY10] H. Wu, Y. Ding, C. Winer, L. Yao. “Network security for virtual machine in cloud computing.” In: *5th International Conference on Computer Sciences and Convergence Information Technology*. Nov. 2010, pp. 18–21. DOI: 10.1109/ICCIT.2010.5711022 (cit. on pp. 25, 34).

[Wil14] W. Williams. *Security for Service Oriented Architectures*. Taylor & Francis Ltd, Apr. 28, 2014. 341 pp. URL: http://www.ebook.de/de/product/23309230/walter_williams_security_for_service_oriented_architectures.html (cit. on pp. 20, 21).

[Win11] J. R. V. Winkler. *Securing the Cloud: Cloud Computer Security Techniques and Tactics*. Syngress Publishing, Apr. 21, 2011. 314 pp. ISBN: 1597495921, 9781597495929 (cit. on pp. 16, 40).

[WLW10] G. Wang, Q. Liu, J. Wu. “Hierarchical Attribute-based Encryption for Fine-grained Access Control in Cloud Storage Services.” In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS ’10. Chicago, Illinois, USA: ACM, 2010, pp. 735–737. ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866414. URL: http://doi.acm.org/10.1145/1866307.1866414 (cit. on p. 30).

[WRLL10] C. Wang, K. Ren, W. Lou, J. Li. “Toward publicly auditable secure cloud data storage services.” In: *IEEE Network* 24.4 (2010), pp. 19–24. DOI: 10.1109/mnet.2010.5510914 (cit. on p. 31).

[WZC+10] L. Wei, H. Zhu, Z. Cao, W. Jia, A. V. Vasilakos. “SecCloud: Bridging Secure Storage and Computation in Cloud.” In: *2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*. Institute of Electrical and Electronics Engineers (IEEE), June 2010. DOI: 10.1109/icdcsw.2010.36 (cit. on p. 32).

[YLL13] C. Yang, W. Lin, M. Liu. “A Novel Triple Encryption Scheme for Hadoop-Based Cloud Data Security.” In: *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*. Institute of Electrical and Electronics Engineers (IEEE), Sept. 2013. DOI: 10.1109/eidwt.2013.80 (cit. on p. 36).

[YRZ09] L. Yan, C. Rong, G. Zhao. “Strengthen Cloud Computing Security with Federal Identity Management Using Hierarchical Identity-Based Cryptography.” In: *Lecture Notes in Computer Science*. Springer Nature, 2009, pp. 167–177. DOI: 10.1007/978-3-642-10665-1_15 (cit. on p. 29).

[ZJRR12] Y. Zhang, A. Juels, M. K. Reiter, T. Ristenpart. “Cross-VM Side Channels and Their Use to Extract Private Keys.” In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS ’12. Raleigh, North Carolina, USA: ACM, 2012, pp. 305–316. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382230. URL: http://doi.acm.org/10.1145/2382196.2382230 (cit. on p. 25).

[ZL12] D. Zissis, D. Lekkas. “Addressing cloud computing security issues.” In: *Future Generation Computer Systems* 28.3 (Mar. 2012), pp. 583–592. DOI: 10.1016/j.future.2010.12.006 (cit. on pp. 26, 41).

[ZWT+14] J. Zhao, L. Wang, J. Tao, J. Chen, W. Sun, R. Ranjan, J. Kołodziej, A. Streit, D. Georgakopoulos. “A security framework in G-Hadoop for big data computing across distributed Cloud data centres.” In: *Journal of Computer and System Sciences* 80.5 (2014), pp. 994–1007 (cit. on pp. 37, 49).

All links were last followed on July 31, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature