Datensicherheit in mandantenfähigen Cloud Umgebungen

Tim Waizenegger[1], Oliver Schiller[1], Cataldo Mega[2]

[1]Universität Stuttgart, Institut für Parallele und Verteilte Systeme
Universitätsstr. 38, 70569 Stuttgart
{Tim.Waizenegger,Oliver.Schiller}@ipvs.uni-stuttgart.de

[2]IBM Software Group
Schönaicherstr. 220, 71032 Böblingen
Cataldo_Mega@de.ibm.com

Abstract: Cloud Computing wird aktuell hauptsächlich für wissenschaftliches Rechnen und endkundenorientierte Dienste verwendet, da die Kostenersparnis hier ein besonders wichtiger Faktor ist. Die Betreiber von Cloud Plattformen sind jedoch immer stärker daran interessiert Cloud Dienste auch im Enterprise Segment anzubieten, um hier gleichermaßen von Kostenvorteilen zu profitieren.

Die Kundenresonanz aus diesem Segment lässt jedoch zu wünschen übrig. Die Gründe dafür sind Bedenken bezüglich Datensicherheit und -vertraulichkeit in mandantenfähigen Systemen. Um diesem Problem zu begegnen, haben wir die Herausforderungen bei der Absicherung von mandantenfähigen Cloud Diensten untersucht, und den Umgang mit vertraulichem Schlüsselmaterial und Anmeldedaten als Schwachstelle identifiziert.

Dieser Beitrag zeigt eine konzeptionelle Lösung zur zentralen Ablage und Zugriffsverwaltung sensibler Daten, sowie deren prototypische Implementierung innerhalb der IBM Cloud Lösung *SmartCloud Content Management*.

1 Einleitung

Cloud Computing zeichnet sich als neues, zukunftsträchtiges Vertriebsmodell für IT ab. Dies wird durch die zunehmende Anzahl existierender Angebote auch im und für das Geschäftsumfeld bestätigt; Gartner sagt voraus, dass 2013 mehr als 80% aller neuen, kommerziellen Anwendungen im Geschäftsumfeld auf Cloud Plattformen betrieben werden [Ga11]. Darüber hinaus wird vorausgesagt, dass bis 2016 40% der Unternehmenskunden eine unabhängige Sicherheitsuntersuchung als Voraussetzung für den Einsatz einer Cloud-Lösung einführen werden. Die zentrale Herausforderung besteht darin, eine ausreichende Sicherheit zu gewährleisten [XF11].

Das erfolgreiche Begegnen dieser Herausforderung wird insbesondere durch eine weitere, für den Erfolg von Cloud Computing essentielle Eigenschaft schwierig: die gemeinsame Nutzung von physischen Ressourcen durch mehrere Kunden [NI11].

Im Geschäftsumfeld entspricht ein Kunde einem Unternehmen, d.h. einer organisatorisch

abgeschlossenen Einheit. In diesem Fall wird der Kunde auch als Mandant bezeichnet. Die Konsolidierung mehrerer Mandanten auf eine physische Ressource wird dementsprechend als *Mandantenfähigkeit* bezeichnet [Wa10]. Mandantenfähigkeit verbessert die Skaleneffekte und verringert die Betriebskosten je Mandant. Dies erhöht zum einem die Gewinnspanne des Betreibers und zum anderen erlaubt es, den Dienst zu attraktiveren Konditionen - verglichen zu konventionellen Lösungen - anzubieten. Mit der Nutzung von Ressourcen durch mehrere Kunden entstehen jedoch zugleich neue Risiken, die zusätzliche Maßnahmen seitens der Cloud Betreiber erfordern, um Datensicherheit zu gewährleisten.

Ein typischer Cloud Dienst ist als Komposition von Programmen aufgebaut welche durch interne Kommunikation einen Mehrwertdienst bilden. Aufgrund dieser Heterogenität ist die Absicherung solcher Systeme alleine durch Einschränkung des Zugriffs schwierig. Falls der physische Zugang zu einem System oder der logische Zugang zu den Daten für Angreifer nicht verhindert werden kann, so bleibt nur die Verschlüsselung der Daten, um unerlaubten Zugriff auszuschließen.

Durch Datenverschlüsselung findet eine Verschiebung des Risikos von den zuvor sensiblen Daten auf das Schlüsselmaterial statt. Dadurch wird eine Reduktion der Menge an kritischen Daten um mehrere Größenordnungen erreicht, mit der die sichere Aufbewahrung dieser Daten erst möglich ist.

Der Einsatz von Verschlüsselung und die hierfür notwendigen Schlüssel machen den Einsatz eines Schlüsselspeichers unerlässlich. Ein solches System muss dafür Sorge tragen, dass ein unerlaubter Zugriff sowie ein Verlust von Schlüsselmaterial verhindert werden. Die verschlüsselten Daten können nur dann als sicher angesehen werden, wenn der Zugriff auf Schlüssel effektiv kontrolliert wird [BEE+10].

Ein wichtiger Aspekt von Cloud Computing ist die Loslösung der Dienste und Daten von physischen Maschinen. Wird daher Datenverschlüsselung in einer Cloudanwendung eingesetzt, so ist der verwendete Schlüssel an die Daten und nicht an die physische Maschine gebunden. Andere physische Maschinen in der Cloud, die zu einem späteren Zeitpunkt auf diesen Daten operieren sollen, benötigen ebenso Zugang zu dem Schlüssel. Damit die verschlüsselten Daten effektiv geschützt sind, muss verhindert werden, dass ein Angreifer, der Zugriff auf die Daten erlangt, auch den Schlüssel erhält. Es ist daher nicht möglich, Daten und Schlüssel gemeinsam abzulegen; ein separater Mechanismus zur Ablage und Verteilung der Schlüssel ist notwendig.

Ein Schlüsselspeicher, der in Cloudanwendungen eingesetzt werden kann, muss Kriterien erfüllen, die über das hinausgehen, was in konventionellen Installationen erforderlich ist. Dynamische Skalierung und Hochverfügbarkeit auf unzuverlässiger Hardware sind Eckpfeiler des Cloud Computings, die wesentlich zum Erfolg des Konzepts beitragen. Ein Schlüsselspeicher muss diese Aspekte daher unterstützen. Kapitel 2.1 zeigt, dass diese Anforderungen in bestehenden Lösungen nur unzureichend erfüllt sind.

Das hier vorgestellte Konzept schließt die Lücke zwischen Schlüsselverwaltung und Cloud Computing und ermöglicht damit den effektiven Einsatz von Verschlüsselung in Cloud Umgebungen.

In Kapitel 2 werden die Basistechnologien aus Cloud Computing und Verschlüsselung eingeführt, sowie ein Überblick über *IBM SmartCloud Content Management* gegeben. SCCM

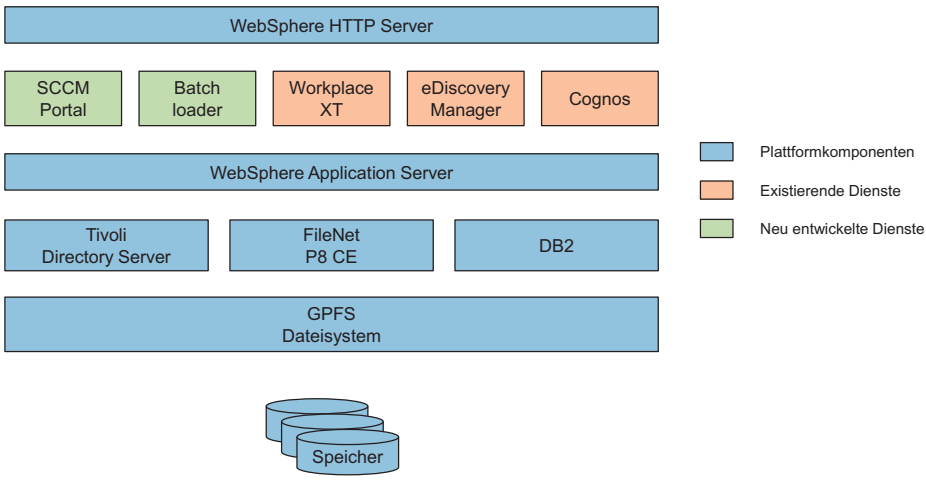

Abbildung 1: Komponenten der SmartCloud Content Management Plattform

ist die Enterprise Cloud-Lösung, anhand derer unsere Untersuchungen stattfanden. Kapitel 3 behandelt im Detail das Konzept und die Architektur unseres Schlüsselspeichers und zeigt dessen Integration in SCCM. In Kapitel 4 bewerten wir unser Konzept anhand der Voraussetzungen des Cloud Umfelds und geben Hinweise auf künftige Arbeiten in Kapitel 5.

2 Hintergrund – IBM SmartCloud Content Management

Das Ziel von *IBM SmartCloud Content Management (SCCM)* ist es, dem Kunden einen Dienst zur Verwaltung sensibler Unternehmensdaten anzubieten, der einen geringen Investitionsaufwand erfordert und günstige Betriebskosten bietet [IB12a]. Diese Randbedingungen legen eine cloudbasierte mandantenfähige Lösung nah, da durch das Wegfallen dedizierter Hardware beim Kunden die Anfangsinvestition niedrig bleibt, während durch Mandantenfähigkeit die Betriebskosten des Anbieters reduziert werden.

SCCM nutzt bestehende IBM Produkte, um die nötige Basisfunktionalität bereitzustellen, sowie neu entwickelte Komponenten wie Mechanismen zum Import von Daten und eine einheitliche Konfigurations- und Verwaltungsoberfläche. Abbildung 1 zeigt den Aufbau von SCCM und gibt einen Überblick über die Komponenten.

Ein solch komponentenbasierter Aufbau ist typisch für Enterprise Cloud Lösungen, da

die Wiederverwendung von etablierten Komponenten im Umfeld von Enterprise-Software, aus Kosten- und Zuverlässigkeitsgründen, der Neuimplementierung vorgezogen wird. Die einzelnen Komponenten eines solchen Systems benötigen jeweils Zugang zu gemeinsamem Schlüsselmaterial, wenn Verschlüsselung eingesetzt wird. Eine weitere Quelle sicherheitsrelevanter Informationen ergibt sich aus der internen Kommunikation der Komponenten, insbesondere der Middleware. Im Fall von SCCM sind dies die DB2 Datenbank und die FileNet P8 Content Engine, deren Dienste von sämtlichen darüber liegenden Komponenten benutzt werden [MKW+09]. Um diese Kommunikation abzusichern, ist die Angabe von Anmeldedaten bei Benutzung der Dienste unerlässlich. Im Zuge der Implementierung einer zentralen Ablage für kryptografisches Schlüsselmaterial wird daher auch die Ablage dieser Anmeldedaten als eine Aufgabe des Schlüsselspeichers angesehen. Der bisherige Ansatz zur Speicherung dieser Daten bestand in der Ablage in Konfigurationsdateien und Konfigurationsdatenbanken, meist im Klartext oder in trivial verschleierter Form [IB12b].

Es besteht daher die Notwendigkeit für ein System, das den Komponenten einer Cloudanwendung sicheren Zugriff auf zentral gespeichertes Schlüsselmaterial ermöglicht. Bei den relevanten Komponenten in SCCM handelt es sich um Anwendungen im Kontext des WebSphere Application Server. Wir erfordern daher ein System, welches sich flexibel an solche Anwendungen anbinden lässt.

2.1 Bestehende Lösungen zur Schlüsselverwaltung

Vor der Entwicklung des Schlüsselspeichers haben wir untersucht, ob eine Neuentwicklung notwendig, oder eine verfügbare Lösung geeignet ist.

Bestehende Verschlüsselungsprodukte setzen stets eine proprietäre Schlüsselverwaltung ein, die nicht von unterschiedlichen Komponenten in einem heterogenen Cloud Umfeld benutzt werden kann. Mit *KMIP*[1] finden aktuell Bemühungen statt, einen solchen Standard zu schaffen. KMIP definiert ein Protokoll zur Kommunikation zwischen Client und Server, welches den gemeinsamen Einsatz von Produkten unterschiedlicher Hersteller ermöglichen soll. Serverseitig unterstützen die im Folgenden vorgestellten Produkte KMIP, jedoch hat kein Hersteller einen KMIP Client im Angebot, der sich flexibel an eigene Anwendungen anbinden lässt.

Die kommerziellen Verschlüsselungsprodukte von *SafeNet*[2] und *Vormetric*[3] bieten eine zentrale Schlüsselverwaltung und die Möglichkeit unterschiedliche Clients anzubinden. Beide Produkte unterstützen jedoch nur eine definierte Menge von Clientanwendungen und können nicht mit selbst entwickelten Anwendungen genutzt werden. Ein Konzept für Mandantenfähigkeit ist ebenfalls nicht vorhanden. Von den Herstellern wird lediglich die Abbildung von Mandanten auf andere strukturelle Objekte in der Schlüsselverwaltung empfohlen.

[1] `https://www.oasis-open.org/committees/kmip`
[2] `http://www.safenet-inc.com`
[3] `http://www.vormetric.com`

Lösungen zum Speichern und Erzeugen von kryptografischem Material sind mit dem Java Key Store und der PKCS12 Implementierung bereits in der Programmiersprache Java enthalten [Or04]. In unserem Prototypen werden diese zum Erzeugen, Signieren und Überprüfen der Zertifikate benutzt, nicht aber, um das Schlüsselmaterial der Clients zu speichern. Der Grund dafür ist die Architektur dieser Key Stores. Sie sind dafür ausgelegt, eine überschaubare Menge von kryptografischem Material zugehörig zu *einer* Entität zu speichern. Weder Anforderungen für Mandantenfähigkeit sind erfüllt, noch solche bezüglich Skalierbarkeit.

Im Gegensatz dazu ist unser Konzept für den Einsatz in Cloud Umgebungen vorgesehen und unterstützt massive Mandantenfähigkeit sowie horizontale Skalierung, wie in Kapitel 4 deutlich wird.

3 Der Schlüsselspeicher für SCCM

Im Folgenden wird das von uns entwickelte Konzept eines Schlüsselspeichers für Cloudanwendungen vorgestellt und anschließend die prototypische Implementierung in SCCM beschrieben. Die zentralen Entwurfsaspekte unseres Konzepts ergeben sich aus den Anforderungen des Cloud Computings. Dies umfasst die funktionalen Aspekte Skalierbarkeit und Hochverfügbarkeit, sowie einen hohen Sicherheitsstandard, der durch die Aufteilung von Zuständigkeiten erreicht wird. Die Zuständigkeiten werden unter drei Parteien aufgeteilt: Einem *Client*, der Zugriff auf Schlüssel beantragt, einem *Server*, der über den Zugriff entscheidet, und einem *Kundenadministrator*, der die Berechtigungen der Clients definiert und verwaltet. Im Folgenden wird mit *Client* eine funktionale Komponente unseres Konzepts bezeichnet, ein Kunde als organisatorische Einheit wird stets *Kunde* genannt.

3.1 Systemarchitektur

Unser Schlüsselspeicher ist als Client-Server Lösung entwickelt. Die Serverkomponente leistet das zentrale Speichern der Schlüssel und die Zugriffsverwaltung, während die Clientkomponente in die Cloudanwendungen integriert ist. Sie implementiert das Kommunikationsprotokoll und bietet der Anwendung über eine Programmierschnittstelle Zugriff auf die Dienste des Servers.

Eine Client-Server-Architektur ist notwendig, da im Cloud-Umfeld verschiedene Instanzen der Anwendung auf denselben Daten operieren und damit Zugriff auf gemeinsames Schlüsselmaterial erfordern. Darüber hinaus werden durch diese Trennung von Zuständigkeiten die sensiblen Schlüssel auf den Server verschoben, was es ermöglicht, zentral über jeden Datenzugriff zu entscheiden.

Als Ergebnis können sämtliche auf dem Client gespeicherten Daten als unkritisch angesehen werden, da ein Angreifer lediglich verschlüsselte Daten vorfindet, jedoch keinen Zugang zu dem Schlüssel hat.

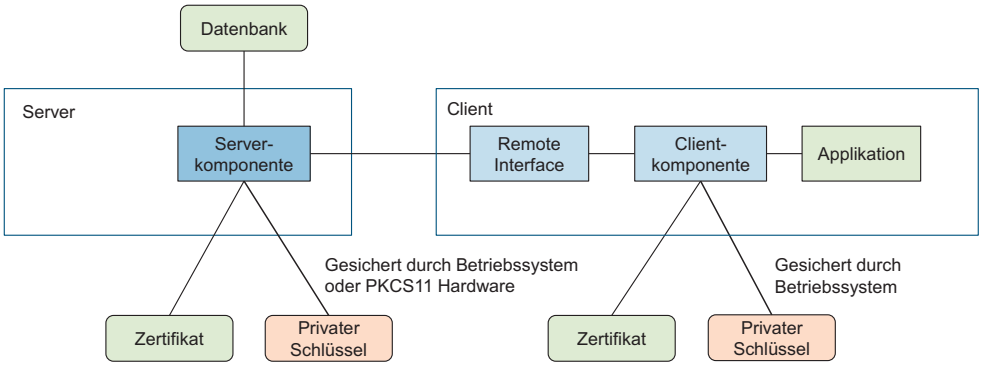

Abbildung 2: Client-Server Architektur des Schlüsselspeichers

Abbildung 2 zeigt die Architektur des Systems. Die Serverkomponente ist an eine relationale Datenbank angebunden, welche das Schlüsselmaterial und die Zugriffsberechtigungen speichert.

Die Cloudanwendung greift über die Programmierschnittstelle der Clientkomponente auf den Server zu, um Schlüsselmaterial zu lesen oder zu schreiben. Für die Anwendung erfolgt dieser Zugriff transparent, da die Clientkomponente Netzwerkkommunikation und Authentifizierung übernimmt.

Falls eine Cloudanwendung konsequent Verschlüsselung mit einem zentralen Schlüsselspeicher einsetzt, so wird dieser zu einer kritischen Komponente, ohne die die gesamte Anwendung nicht funktionieren kann. Daher war bei der Entwicklung unseres Konzepts die Ausfallsicherheit und Redundanz der Serverkomponente ein wichtiges Kriterium. Um auf bestehende, etablierte Technologien zurückgreifen zu können, ist die Serverkomponente zustandslos ausgelegt. Dies wird umgesetzt mit einer anfragebasierten Authentifizierung. Damit muss der Server keine Sitzungen verwalten und kann jede Anfrage unabhängig von den vorhergehenden bearbeiten. Die Anfragen können daher beliebig auf ein verteiltes Cluster von Anwendungsservern verteilt werden. Zur Ablage des Schlüsselmaterials wird eine relationale Datenbank benutzt, welche durch Replikation oder Partitionierung ebenfalls verteilt ausgeführt werden kann.

In großen Cloud-Umgebungen mit hohem Lastaufkommen erlauben diese Designaspekte neben der Ausfallsicherheit, durch horizontale Skalierung die Leistung des Systems auf-

recht zu erhalten.

Der Schlüsselspeicher wurde für die Anwendung in SCCM - einer Enterprise-Lösung - entwickelt. Mandantenfähigkeit bedeutet in diesem Kontext, dass sich mehrere Kunden zwar die physische Maschine teilen, die darüber liegende Software aber zu genau einem Kunden gehört. Der Server ist daher in der Lage, Clients zu bedienen, die zu verschiedenen Kunden gehören. Die Clientkomponente ist nicht mandantenfähig ausgelegt, sondern an einen Kunden gebunden. Sie weist die Zugehörigkeit zu einem Kunden während der Authentifizierung dem Server gegenüber aus, sodass ein Zugriff auf fremdes Schlüsselmaterial ausgeschlossen ist. Das Kommunikationsprotokoll basiert auf gegenseitiger Authentifizierung mit asymmetrischer Verschlüsselung.

Sowohl die Server- als auch die Clientkomponente benötigen ein Zertifikat und den dazugehörigen privaten Schlüssel. Der private Schlüssel ist das Gegenstück zu dem öffentlichen Schlüssel in dem Zertifikat. Die Rolle der Zertifikate und deren Erstellung wird in Kapitel 3.2.1 behandelt. Der private Schlüssel stellt das letzte Glied in einer Kette von Risikoverschiebungen dar, weshalb seine Sicherheit von kritischer Bedeutung ist. Der private Schlüssel auf Seite des Servers wird benutzt, um den Inhalt der Datenbank zu verschlüsseln. In unserer Lösung ist er daher in einem Hardware Key-Store abgelegt, aus dem ihn nur der legitime Serverprozess auslesen kann [Gu09]. Der private Schlüssel des Clients wird nicht zur Verschlüsselung von Daten verwendet, sondern dient dem Client dazu sich gegenüber dem Server zu authentifizieren. Er wird mit Betriebssystemmitteln wie SELinux vor unerlaubtem Zugriff geschützt [KAAS11].

Um die Sicherheit der Daten zu gewährleisten, darf der Schlüsselspeicher nur Zugriff auf Schlüssel gewähren, falls die Anfrage von einer Clientkomponente stammt, die die nötige Berechtigung aufweist. Im Folgenden wird daher beschrieben, wie der Server das Schlüsselmaterial ablegt und mit einem neuartigen Autorisierungsschema die Clientberechtigungen zuordnet.

3.2 Autorisierungsschema – Zugriffsabstraktion über Gruppen

Das Autorisierungsschema beschreibt die Datenstrukturen und Prozesse, durch welche die Berechtigungen der Clients den Objekten in dem Schlüsselspeicher zugeordnet werden.

Das Entity-Relationship Modell in Abbildung 3 zeigt die Datenobjekte, die das Autorisierungsschema ausmachen. Es kann anhand der enthaltenen Beziehungen direkt in ein relationales Datenmodell für den Server umgesetzt werden. Die drei wesentlichen Objekte in dem Schema sind der Client, die Gruppe und der Schlüssel.

Ein *Client* im Modell des Autorisierungsschemas ist eine Instanz der im vorigen Abschnitt beschriebenen Clientkomponente, die auf mindestens einem Cloud-Rechner aktiv ist. Der *Schlüssel* ist das Datenobjekt, welches das sensible Schlüsselmaterial enthält und über die *Gruppe* mit den Clients verknüpft ist, denen Zugriff gewährt werden soll. Die Kardinalitäten der Objektbeziehungen in Abbildung 3 verdeutlichen die Funktion der Gruppen als zentralen Aspekt des Autorisierungsschemas. Jeder Schlüssel ist genau einer Gruppe zugeordnet, ein Client wiederum kann Mitglied beliebig vieler Gruppen sein. Ebenso kann

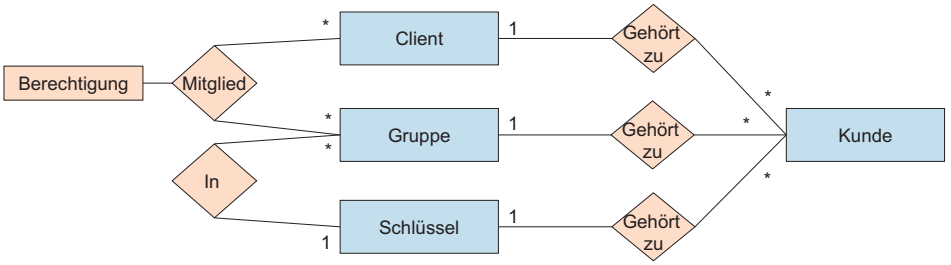

Abbildung 3: Zugriffskontrolle über Gruppen

jede Gruppe beliebig viele Mitglieder haben.

Die Beziehung, welche die Gruppenmitgliedschaft ausdrückt, hat ein Attribut welches die Art der Berechtigung kennzeichnet, die der Client auf Schlüssel dieser Gruppe hat. In unserem Konzept sind drei Mitgliedschaftstypen umgesetzt: *schreibend*, *lesend* und *hinzufügend*. Der erste Typ erlaubt vollständigen Zugriff, der das Hinzufügen, Löschen und Lesen von Schlüsseln ermöglicht. Die letzteren Typen schränken diesen Zugriff entsprechend ein. Der Typ *hinzufügend* kann dabei wie ein Nachttresor verstanden werden, in den lediglich neues Schlüsselmaterial eingefügt, jedoch kein bestehendes manipuliert oder gelesen werden kann.

Mit diesem Autorisierungsschema ist es dem Server möglich, über Zugriffe zu entscheiden, ohne Wissen über den Client vorzuhalten. Der Server speichert lediglich die Zuordnung der Schlüssel zu Gruppen, entsprechend obigem Schema. Um über den Zugriff eines Clients zu entscheiden, benötigt der Server dessen Gruppenmitgliedschaft. Im Folgenden wird die Methode beschrieben, durch welche ein Client dem Server gegenüber die Berechtigungen ausweisen kann, welche der Kundenadministrator ihm zugewiesen hat.

3.2.1 Zertifikatbasierte Autorisierung der Clients

Die Verwendung von Zertifikaten zur *Authentifizierung* hat eine lange Tradition und gilt anderen Konzepten wie der passwortbasierten Authentifizierung, aufgrund der Länge des geheimen Datums, als überlegen [Am94].

Um die Interaktion mit der Serverkomponente zu minimieren und die Aufteilung von Zuständigkeiten umzusetzen, wurde im Rahmen dieses Beitrags die Verwendung von Zertifikaten auf die *Autorisierung* ausgeweitet. Dieser Ansatz macht Gebrauch von erweiterten Attributen in Zertifikaten, mit denen es möglich ist, abgesehen von den Basisinformationen zur Identifikation, zusätzliche Daten in dem signierten Teil des Zertifikats abzulegen [RSA83].

Wir verwenden diese Datenfelder für das Speichern von Zugriffsberechtigungen, wobei die Signatur des Zertifikats gewährleistet, dass jede Manipulation entdeckt wird. Dabei sind die anfangs erwähnten drei Parteien beteiligt. Das Zertifikat wird von einem *Kunde-*

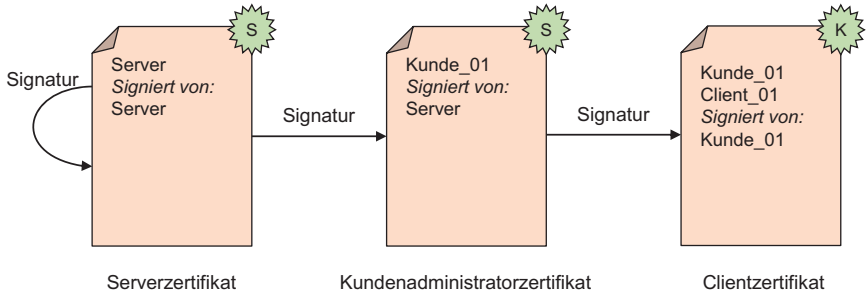

Abbildung 4: Beziehungen der Zertifikate

nadministrator überprüft und signiert und dem *Client* übergeben, der das Zertifikat benutzt, um seine Berechtigungen gegenüber einem *Server* auszuweisen. Das Vertrauensverhältnis zwischen Kundenadministrator und Server wird auf den Client erweitert, da der Server einem Zertifikat vertraut indem er die Legitimität der Signatur überprüft, womit er sich von der Echtheit des Zertifikats überzeugt hat [An01].

3.3 Aufteilung der Zuständigkeiten durch Zertifikatshierarchie

Im Folgenden wird beschrieben, wie das Autorisierungsschema aus dem vorigen Kapitel mittels zertifikatbasierter Autorisierung umgesetzt wird, und in welcher Beziehung die beteiligten Zertifikate stehen.

Um die Zuordnung von Gruppenberechtigungen auf Clients mit Zertifikaten umzusetzen, wird für jeden Client ein Zertifikat ausgestellt. Dieses wird stets einem Client, also einer Instanz der Clientkomponente, zugeordnet und enthält die Gruppenmitgliedschaften und Berechtigungen dieses Clients.

Die nötige Hierarchie von Zertifikaten ist dargestellt in Abbildung 4. Sie beginnt mit dem Server, welchem bei der Installation des Systems ein selbst-signiertes Zertifikat zugewiesen wird. Dieses Serverzertifikat wird verwendet, um das Kundenadministratorzertifikat bei der Einführung eines neuen Kunden zu signieren. In diesem Prozess behält der Server eine Kopie des ausgestellten Kundenzertifikats, mit dessen Hilfe er die Gültigkeit späterer Clientzertifikate überprüft [CA61]. Der Kundenadministrator ist nun in der Lage die Cloudanwendung zu provisionieren und Clientzertifikate mit Gruppenzugehörigkeit auszustellen.

Mit dieser Aufteilung der Zuständigkeiten ist der Herausgeber der Zertifikate - also der Kundenadministrator - dafür verantwortlich, nur legitimen Clients Zugriff zu gewähren.

3.4 SCCM Integration

Zur Evaluation unseres Konzepts wurde ein Prototyp in Java implementiert, dessen Clientkomponente als Dienst von den Komponenten der Cloudanwendung SmartCloud Content Management benutzt wird. Als Integrationsebene wurden die Anwendungen des WebSphere Application Server gewählt, da diese verteilt auf unterschiedlichen Maschinen laufen und verschiedene Aufgaben erfüllen, was eine Zuordnung von disjunkten Zugriffsrechten ermöglicht. Jede dieser Applikationen aus Abbildung 1 wird daher als Client behandelt und erhält ein eigenes Zertifikat mit den Zugangsberechtigungen, die für diese Applikation erforderlich sind.

Wie in Kapitel 1 beschrieben, soll der zentrale Schlüsselspeicher sowohl kryptografisches Schlüsselmaterial, als auch Anmeldedaten zu internen Systemen absichern. eine Clientkomponente für unseren Schlüsselspeicher wird daher an die Anwendungen *SCCM Portal*, *Batch loader*, *WorkplaceXT* und *eDiscovery Manager* angebunden. Für jede dieser Anwendungen wird ein Clientzertifikat ausgestellt, das die nötigen Zugriffsberechtigungen widerspiegelt.

Die in Kapitel 3.2 eingeführte parametrisierte Gruppenmitgliedschaft macht es nun möglich, den Batch loader auf einfügenden Zugriff zu beschränken, wenn er neue Schlüssel für geladene Daten speichert. Die Komponente eDiscovery Manager dient dem Finden und Abrufen von Daten und erhält daher lesenden Zugriff, während WorkplaceXT schreibenden Zugriff erfordert. Das Cognos Berichterstellungssystem ist in die Benutzeroberfläche des SCCM Portals integriert und benötigt an dieser Stelle nur lesenden Zugriff auf die DB2 Datenbank, weshalb hier ein Clientzertifikat vergeben wird, das lediglich Zugang zu den Anmeldedaten für einen eingeschränkten DB2 Benutzer ermöglicht.

4 Diskussion und Evaluation

Um die Eignung des hier vorgestellten Konzepts in einem Cloud Umfeld zu beurteilen, werden im Folgenden die Skalierbarkeit, Hochverfügbarkeit und Sicherheit anhand von Aspekten der Systemarchitektur gezeigt.

Hochverfügbarkeit wird mit horizontaler Skalierung erreicht. Beim Ausfall von Servern stehen weitere zur Verfügung, auf welche die Anfragen umgeleitet werden. Derselbe Mechanismus erlaubt es, bei großer Last die Anfragen parallel auf verteilten Servern auszuführen und damit Engpässe zu vermeiden.

Zwei wichtige Eigenschaften der Architektur ermöglichen eine horizontale Skalierung der Serverkomponente über ein Cluster von Anwendungs- und Datenbankservern. Dies sind die Zustandslosigkeit der Serverkomponente, sowie die Datenhaltung in einer relationalen Datenbank. Damit ist es möglich, horizontale Skalierung mit Funktionen der Middlewarekomponenten umzusetzen anstatt eine anwendungsspezifische Lösung zu erfordern.

Auf Seite des Anwendungsservers kann die Serverkomponente, ebenfalls wegen der Zustandslosigkeit, in beliebig vielen Instanzen ausgeführt werden. Ein Load Balancer ver-

teilt die Anfragen der Clients unter diesen. Unsere Implementierung setzt dieses Konzept durch den Einsatz eines WebSphere Clusters um. Dieses besteht aus einer Anzahl von Servern, die auf separaten physischen Maschinen laufen und jeweils eine Instanz der Schlüsselspeicher Serverkomponente ausführen. Der WebSphere Cluster wird verwaltet von einem zentralen Deployment Manager. Dieser koordiniert das Hinzufügen und Entfernen von Servern innerhalb des Clusters und dient als Load Balancer welcher Clientanfragen auf die Clustermitglieder verteilt[4]. Um die Verfügbarkeit des Systems auch bei Ausfall des Deployment Managers zu gewährleisten, wird dieser in einer active-passive Konfiguration betrieben. Eine zweite passive Instanz des Deployment Managers läuft dabei auf einer separaten physischen Maschine und synchronisiert ihre Konfiguration mit dem aktiven Deployment Manager. Sollte dieser nicht mehr erreichbar sein, wird die passive Instanz aktiv und nimmt die Anfragen entgegen.

Im Gegensatz zu dem Anwendungsserver sind die Instanzen der relationalen Datenbank nicht unabhängig voneinander, da sie einen konsistenten Datenbestand repräsentieren müssen. Es ist daher eine Synchronisation der Instanzen notwendig. Die Synchronisation der Datenbankinstanzen kann mit unterschiedlichen Verfahren umgesetzt werden.

In unserem Konzept wurde die Skalierung der Datenbank durch Replikation und Partitionierung gelöst[5]. Besonders in Szenarien, die nur wenige schreibende Zugriffe auf den Schlüsselspeicher erfordern, ist Replikation gut geeignet, da sich eine konsistente Kopie des Datenbestandes auf jedem Server befindet. Die Anfragen der Anwendungsserver können daher beliebig auf die Datenbankserver verteilt werden, was Engpässe vermeidet.

In Szenarien, die viele Schreibzugriffe auf verteilte Datenbankserver erfordern, ist vollständige Replikation jedoch schlecht geeignet, da zur Erhaltung der Konsistenz eine teure Synchronisation der Datenbankserver erforderlich ist. Eine bessere Strategie für solche Szenarien ist Datenbankpartitionierung, da sie keine Synchronisation der verteilten Datenbanken erfordert.

Wir verwenden daher einen heterogenen Ansatz aus Partitionierung und Replikation, um die Vorteile beider Mechanismen auszunutzen. Wir verwenden horizontale Partitionierung auf dem Schlüsselmaterial, während die Partitionen auf mehrere Server repliziert werden. Mit derzeitigen Systemen ist nur statische Partitionierung möglich, so dass zum Zeitpunkt der Systemprovisionierung entschieden werden muss, anhand welcher Kriterien die Aufteilung erfolgt. Im Kontext von Enterprise-Anwendungen ist es sinnvoll nach Mandanten zu partitionieren, da so jedem Kunden dedizierte Ressourcen zugesichert werden können. Die Datenpartition eines Kunden wird durch Replikation auf mehrere Server verteilt, um den jeweiligen Anforderungen des Kunden nachzukommen.

Der Grund für die Entwicklung des zentralen Schlüsselspeichers sind Defizite in bestehenden Lösungen. Sie sind nicht mandantenfähig und lassen sich nicht an eigene Anwendungen anbinden. Diese Anforderungen wurden in unserem Konzept berücksichtigt. Mit der Implementierung unseres Prototyps basierend auf einer relationalen Datenbank und einer zustandslosen Serverkomponente werden die, in Cloud Umgebungen wichtigen, Anforde-

[4] http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/welcclusters.html

[5] http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.swg.im.iis.db.repl.sqlrepl.doc/topics/iiyrscncsqlreplovu.html

rungen an Skalierung und Hochverfügbarkeit auf etablierte Systeme delegiert.

Die Einführung der zertifikatbasierten Autorisierung ermöglicht es in Cloud Umgebungen Vertrauen beim Kunden zu schaffen, indem dedizierte Kundenadministratoren bestimmt werden, die alleinig über die Zugriffsberechtigungen auf das Schlüsselmaterial entscheiden.

Sicherheit vor Betrug durch die teilnehmenden Parteien ist in diesem System implizit durch die Integrität der Zertifikate gegeben. Ein Kundenadministrator hat legitime Kontrolle über alle Berechtigungen unterhalb seiner Stufe, er kann aber nicht durch Manipulation die Daten anderer Kunden kompromittieren. Der Server kann die Zertifikate, die ein Kundenadministrator ausstellt, eindeutig diesem Kunden zuordnen, selbst wenn der Administrator die Identitätsinformationen des Zertifikats manipuliert. Dasselbe gilt für die Clientzertifikate. Durch die Signatur sind sie vor Manipulation durch jemanden anderen als den Kundenadministrator geschützt.

Unser Konzept der zentralen Schlüsselverwaltung mit verteilten Zuständigkeiten ermöglicht es daher, in einer Cloud Umgebung einen höheren Sicherheitsstandard zu erreichen. Das Risiko wird verschoben von den Rechenknoten in der Cloud auf eine kleine Anzahl von Servern. Durch die neue Rolle des Kundenadministrators wird darüber hinaus - im Cloud Umfeld vermisstes - Vertrauen an den Kunden zurückgegeben und eine attraktive Cloud Plattform geschaffen.

5 Ausblick

Der Kern dieses Beitrags ist die Trennung von Zuständigkeiten zwischen dem Client, welcher kryptografisches Schlüsselmaterial erzeugt und benutzt, und dem Server der es verwaltet. Damit ist die situationsabhängige Autorisierung von Clients möglich, womit die Grundlage geschaffen wurde, mit einem Überwachungssystem kompromittierte Clients zu erkennen, und deren Zugriff zu sperren.

In zukünftigen Arbeiten muss daher untersucht werden, mit welchen Mechanismen eine solche Erkennung möglich ist, und wie sie vor Manipulation geschützt werden kann.

Viele Rechenzentren verfügen über Sicherheitssysteme, die mit Sensoren den unerlaubten Zugang zu Räumen und Maschinen erkennen. Leicht können diese Informationen heran gezogen werden, um kompromittierte Maschinen zu identifizieren.

Literatur

[Am94] Edward Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994.

[An01] Ross Anderson. *Security Engineering*. John Wiley & Sons, Chichester, 2001.

[BEE+10] Jean Bacon, David Evans, David M. Eyers, Matteo Migliavacca, Peter Pietzuch und Brian Shand. Enforcing end-to-end application security in the cloud (big ideas paper).

In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, Seiten 293–312, Berlin, Heidelberg, 2010. Springer-Verlag.

[CA61] Steve Lloyd Carlisle Adams. *Understanding Pki: Concepts, Standards, and Deployment Considerations*. Pearson, 1961.

[Ga11] Gartner. Gartner Reveals Top Predictions for IT Organizations and Users for 2012 and Beyond. 2011.

[Gu09] PKCS 11 Base Functionality v2.30: Cryptoki, July 2009.

[IB12a] IBM. Cloud-based content management service. Bericht, 2012.

[IB12b] IBM. IBM WebSphere Application Server V8.5 information center - Java 2 Connector authentication data entry settings, 2012.

[KAAS11] K.A. Khan, M. Amin, A.K. Afridi und W. Shehzad. SELinux in and out. In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, Seiten 339 –343, may 2011.

[MKW+09] Cataldo Mega, Kathleen Krebs, Frank Wagner, Norbert Ritter und Bernhard Mitschang. *Content-Management-Systeme der nächsten Generation*, Seiten 539–567. Wissens- und Informationsmanagement; Strategien, Organisation und Prozesse. Gabler Verlag, Wiesbaden, January 2009.

[NI11] National Institute of Standards und Technology. The NIST Definition of Cloud Computing. *Special Publication 800-145*, 2011.

[Or04] Oracle. *Java PKCS11 Reference Guide*, 2004.

[RSA83] R. L. Rivest, A. Shamir und L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26:96–99, January 1983.

[Wa10] Phil Wainewright. Defining the true meaning of cloud. *ZDNet*, 2010.

[XF11] IBM X-Force. IBM X-Force® 2011 Mid-year Trend and Risk Report. Bericht, IBM Security Solutions, 2011.