Implementierung von kryptographischen Sicherheitsverfahren für Apache Cassandra und Apache HBase

Autoren: Tim Waage und Lena Wiese[1], Research Group Knowledge Engineering, Institut für Informatik, Georg-August-Universität Göttingen

Abstract:

Spaltenfamiliendatenbanken (engl.: „column family databases" oder „wide column stores") sind wegen ihres flexiblen Datenmodells beliebt, das eine weitgehend schemalose Datenverwaltung ermöglicht. Diese Datenbanken (insbesondere HBase und Cassandra als quelloffene Produkte) werden auch von einigen großen Cloud-Dienstanbietern als Database-as-a-Service bereitgestellt. Eine Verschlüsselung der Transportschicht (also eine Sicherung der Verbindung zwischen Kundenrechner und der Cloud-Datenbank) ist in der Regel vorgesehen. Jedoch wird eine darüberhinausgehende Verschlüsselung der Daten innerhalb der Datenbank entweder gar nicht oder zu spät (d.h. erst auf Datenbankseite) unterstützt. Die Daten sind daher im Klartext zugreifbar für den Dienstanbieter. Im Falle eines Einbruchs in das Datenbanksystem kann darüber hinaus auch ein externer Angreifer vollen Zugriff auf die Daten erhalten.

Verschlüsselung ist daher notwendig, um Daten vor unbefugtem Zugriff zu schützen. Insbesondere sollte die gesamte Schlüsselverwaltung sowie die Ver- und Entschlüsselungsoperationen auf Kundenseite erfolgen, damit die Vertraulichkeit der Daten gewahrt bleibt. Traditionelle starke Verschlüsselungsverfahren führen jedoch dazu, dass die Daten nicht mehr effizient verwaltet werden können: eine Suche nach übereinstimmenden Werten, eine Sortierung oder eine Aggregation (zum Beispiel Summierung) der Daten ist nicht möglich. Zur Lösung dieser Probleme wurden in der Theorie zahlreiche Verschlüsselungsverfahren vorgeschlagen, die gewisse Eigenschaften des Klartextes erhalten (sogenannte durchsuchbare und ordnungserhaltende Verschlüsselung).

In diesem Artikel behandeln wir zum einen die derzeit vorhandenen Angebote, HBase und Cassandra als Database-as-a-Service zu nutzen und zum Anderen stellen wir unsere Implementierungen von Verschlüsselungsverfahren vor, die es ermöglichen, Cloud-Datenbanken (und zwar speziell HBase und Cassandra) mit verschlüsselten Daten zu nutzen.

Stichworte: Spaltenfamiliendatenbanken, Cloud-Datenbanken, HBase, Cassandra, Kryptographie

[1] [tim.waage|wiese]@informatik.uni-goettingen.de, Institut für Informatik, Goldschmidtstraße 7, 37077 Göttingen

1. Spaltenfamilien-Datenbanken

Spaltenfamilien-Datenbanken sind eine Untergruppe von NoSQL Datenbanken (für eine Übersicht siehe beispielsweise [Wiese15]), die durch Googles BigTable-System [Chang06] angeregt wurden. Zu den als freie und quelloffene Software verfügbaren Spaltenfamilien-Datenbanken gehören Apache Cassandra [Lakshman10], Apache HBase (als Teil des Hadoop-Projektes [Borthakur11]), Apache Accumulo [Byun12] und Hypertable. Spaltenfamilien-Datenbanken zeichnen sich durch eine Reihe positiver Eigenschaften aus, die wir hier kurz zusammenfassen. Sie implementieren das Konzept der Spaltenfamilie, die jeweils eine Teilmenge der Spalten zusammenfasst. Tabellenzellen werden durch Schlüssel-Wert-Paare repräsentiert. Genauer gesagt besteht der Schlüssel aus einem Zeilenidentifikator (engl.: „row id“), einem Spaltenfamiliennamen (engl.: „column family name“), einem Spaltenbezeichner (engl.: „column qualifier“) und einem Zeitstempel (engl.: „timestamp“). Die Benutzung von Zeitstempeln ermöglicht eine automatische Versionierung der gespeicherten Daten. Die Versionierung anhand von Zeitstempeln kann so konfiguriert werden, dass nur eine festgelegte Höchstanzahl von Versionen gespeichert werden (die ältesten werden dann automatisch gelöscht). Alternativ kann beim Einfügen eines Schlüssel-Wert-Paares gleich eine maximale Lebensdauer (engl.: „time-to-live“) angegeben werden, nach deren Ablauf das Schlüssel-Wert-Paar gelöscht wird.

Während Spaltenfamilien-Datenbanken zwar auch Tabellen, Zeilen und Spalten besitzen wie traditionelle relationale SQL Datenbanken, ist der fundamentale Unterschied jedoch, dass Spalten individuell für jede Zeile definiert werden können und nicht von einer Tabellenstruktur vorgegeben werden. So können zwei Tabellenzeilen vollkommen verschiedene Spalten (mit jeweils anderen Bezeichnern) enthalten. Dies ist ein klarer Gegensatz zu einem relationalen Tabellenschema, das eine feste Menge von Spalten für alle Zeilen festlegt. Das bedeutet insbesondere, dass nicht jede Tabellenzeile Werte für alle Spalten enthalten muss. Spaltenfamilien-Datenbanken unterstützen damit schwach besetzte Tabellen besser: während nicht vorhandene Werte im relationalen Modell durch NULL dargestellt werden, ist bei Spaltenfamilien-Datenbanken das entsprechende Schlüssel-Wert-Paar einfach nicht vorhanden.

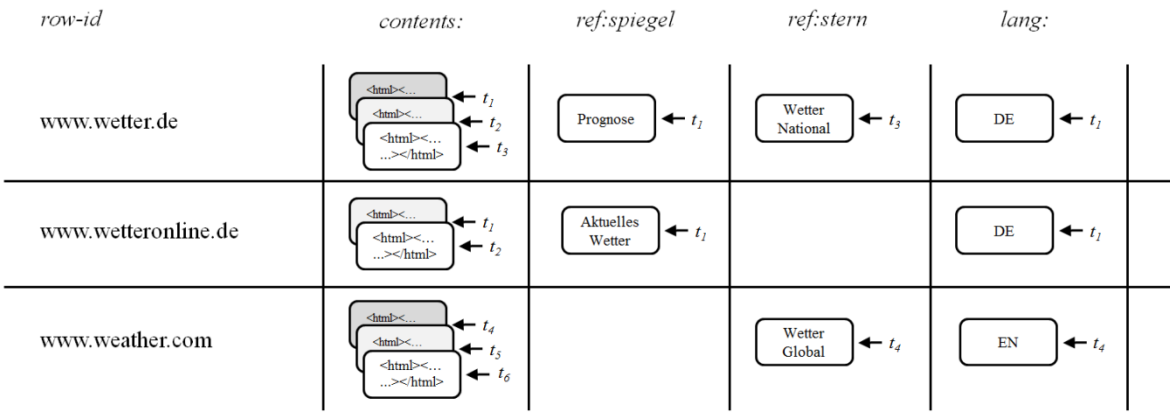


Abb. 1: Eine Beispieltabelle, die verschiedene Wetterseiten speichert und die Art, wie Nachrichtenseiten auf sie verlinken. Eindeutige Row-id ist die URL der Wetterseite. Die „contents“ Spalte enthält durch einen Zeitstempel *t* versionierte Snapshots der Webseiten. Es handelt sich darüber hinaus um eine nicht vollständig besetzte Tabelle. Der Spiegel verlinkt beispielsweise nicht auf weather.com, der entsprechende Tabelleneintrag existiert somit gar nicht. Die Spaltenfamilie „ref“ enthält zwei Spalten, alle anderen Spaltenfamilien in diesem Beispiel nur eine.

Unser Projekt konzentriert sich mit Apache Cassandra und Apache HBase auf die populärsten Vertreter der Kategorie der Spaltenfamilien-Datenbanken[2]. Beide sind weit verbreitet und werden aktiv weiterentwickelt. Sie basieren in den Grundzügen ihrer Datenmodelle auf sehr ähnlichen Konzepten, verfolgen aber jeweils vollkommen unterschiedliche architektonische Ansätze, die wir im Folgenden kurz beschreiben.

Apache Cassandra bietet mit der Cassandra Query Language (CQL) eine Abfragesprache, die sehr stark an SQL angelehnt ist. Aufgrund der Schemafreiheit des zugrundeliegenden Datenmodells ist sie jedoch nicht ganz so umfangreich. Es existieren zahlreiche Treiber, die CQL für diverse verbreitete Programmiersprachen wie Python, C#, .NET, Java, Ruby u.a. nutzbar macht. Innerhalb des Projektes wird in Java entwickelt. Cassandra folgt dem Peer-to-Peer Prinzip, d.h. alle Knoten eines Clusters werden gleich behandelt. Damit werden bzgl. des CAP-Theorems [Brewer10] besonders Hochverfügbarkeit und Partitionstoleranz bedient. Pro Knoten läuft Cassandra in einem einzigen Java Prozess.

Apache HBase bringt eine native Java API mit, die im Projekt verwendet wird. Es basiert auf Apache Hadoop, einem Java Framework zur Verarbeitung großer Datenmengen auf Serverclustern und verfolgt einen anderen Ansatz als Cassandra. HBase ist in erster Linie mehr als Data Store zu verstehen, denn es bringt weniger Funktionalität mit, als man es von klassischen relationalen Datenbanken und auch einigen anderen NoSQL-Datenbanken gewohnt ist. Statt einer eigenen Anfragesprache gibt es im Wesentlichen nur vier Operationstypen (get, put, scan und delete) zur Interaktion mit der Datenbank. Darüber hinaus wird nicht unterschieden zwischen verschiedenen Datentypen, wie beispielsweise Integer für Zahlen oder Strings für Text. Stattdessen wird alles als Byte-Array behandelt. Das verleiht HBase die Fähigkeit mit extrem großen Datenmengen mit Millionen oder sogar Milliarden Datensätzen umzugehen. HBase basiert auf einer Master-Slave-Architektur. Eine zusätzliche Software („Zookeeper“) koordiniert, konfiguriert und synchronisiert verschiedene HBase-Prozesse, von denen mehrere pro Knoten ausgeführt werden können.

2. Datensicherheit in der Cloud

Einhergehend mit der starken Zunahme der Menge zu speichernder Daten gibt es vermehrt Cloud-Angebote auf dem Markt, mit denen sich Kunden Speicherplatz bei einem externen Anbieter mieten können. Dabei werden gerade Spaltenfamilien-Datenbanken wegen ihres flexiblen Datenmodells gerne von Cloudspeicher-Anbietern unterstützt.

2.1. Übersicht

International führende Cloudspeicher-Anbieter sind beispielsweise Microsoft Azure, Google Cloud Platform, Amazon Web Services und Rackspace. Sie unterscheiden sich in erster Linie durch Preis und Leistungsspektrum. Da ein „manuelles“ Aufsetzen von Cassandra und HBase Clustern zeitaufwendig und kompliziert ist, bieten sie aber auch diverse Mechanismen, die ein entsprechendes Deployment vereinfachen können. Diese werden im Folgenden kurz erläutert.

Die *Google Cloud Platform* bringt bereits viel Funktionalität mit, um Cassandra zu betreiben. Ein Cluster mit beliebig vielen Nodes und auf Hardware verschiedener Leistungsklassen lässt sich einfach über ein Web-Interface („Cloud Launcher“) erstellen. Googles Cloud SDK bietet dann diverse Tools, um vorwiegend SSH-basiert mit dem Cassandra Cluster zu interagieren. Das gleiche Prozedere ist anzuwenden für die Erstellung eines Hadoop Clusters, der als Grundlage für HBase benötigt wird.

Microsoft Azure bietet zunächst keine native Unterstützung für zustandslose Datenbanken wie Cassandra. Drittanbieter (z.B. Instaclustr) machen die Erstellung von Cassandra Clustern aber ähnlich einfach wie unter der

[2] Vgl. http://db-engines.com/de/ranking, besucht am 24.03.2016

Google Cloud Platform. Im Gegensatz dazu findet Apache HBase von Haus aus native Unterstützung mit einem Service zur Verwaltung von Hadoop-basierten Projekten („HDInsight“), der ein entsprechendes Web Interface zur Verfügung stellt.

Die *Amazon Web Services* (Amazon EC2) bieten keine native Unterstützung für Cassandra. So genannte Amazon Machine Images (AMIs) von Drittanbietern vereinfachen die Erstellung eines Clusters jedoch immens. Diese werden über ein Web Interface („AWS Console“) auf der gewünschten Hardware installiert und stellen dann ihrerseits ein weiteres Web Interface („OpsCenter“) zur Verfügung, das zur konkreten Erstellung und dem Monitoring von Cassandra Clustern benutzt werden kann. Für das Betreiben von HBase Clustern auf Amazons EC2 Plattform sind ebenso einige Vorarbeiten nötig. AWS bietet einen Elastic MapReduce Service („Amazon EMR“), mit dem über ein Web Interface wiederum Apache Hadoop Cluster erstellt werden können. Amazon stellt für die Installation ähnlich zum Prozedere bei Cassandra diverse AMIs zur Verfügung.

Rackspace bietet das Hosting von Cassandra Clustern als „managed service“ an. Intern wird dabei „Ansible“ benutzt, eine Open-Source-Plattform zur allgemeinen Konfiguration und Administration von Computern. Für die Erstellung von HBase Clustern bietet Rackspace in Kooperation mit Drittanbietern spezielle Tools an. Beide Datenbanken können aber auch mit Apache Whirr eingerichtet werden, einer Sammlung von Bibliotheken der Apache Foundation zur Verwaltung von Cloud Services, die jedoch seit 2015 nicht mehr aktiv weiterentwickelt wird.

2.2. Ziele

Unser Ziel ist die Entwicklung einer Plattform, um Daten in verschlüsselter Form bei verschiedenen Clouddatenbank-Anbietern speichern zu können. Dabei sollen die Daten sowohl vor „Angriffen“ durch einen ehrlichen aber neugierigen (engl.: „honest-but-curious“) Cloudspeicher-Anbieter, aber auch vor möglicherweise böswilligen Angriffen Dritter geschützt werden. Eine besondere Schwierigkeit besteht darin, dass je nach Clouddatenbank unterschiedliche Zugriffsmethoden und Datenmodelle benutzt werden.

Die größte Herausforderung ist es, die technischen Interna der Datenbanken unverändert zu belassen, da auf Seiten der Cloud-Anbieter keine Änderung der bestehenden Installationen möglich ist bzw. vermieden werden soll. Auch die Sortierung von Schlüsseln und die effiziente Anfragebearbeitung sollen weiterhin wie mit unverschlüsselten Daten funktionieren. Dazu mussten diverse kryptographische Verfahren implementiert und für den Einsatz mit Clouddatenbanken optimiert werden.

Unser Lösungsansatz besteht daher darin, eine zusätzliche Software (einen Proxy-Client) zu entwickeln, die in einer vertrauenswürdigen Umgebung (in der Regel also auf Nutzerseite) ausgeführt wird und dort die Schlüsselverwaltung sowie die Ver- und Entschlüsselung übernimmt. Sie wird also – wie in Abb. 1 dargestellt – zwischen den Benutzer und die Clouddatenbank geschaltet. Pro Datenbank ist ein Proxy-Client notwendig, zu dem sich aber mehrere Nutzer verbinden können. Ein Proxy-Client könnte auch die Verwaltung mehrerer Datenbanken übernehmen, wobei er dann aber auf entsprechend performanter Hardware laufen sollte.

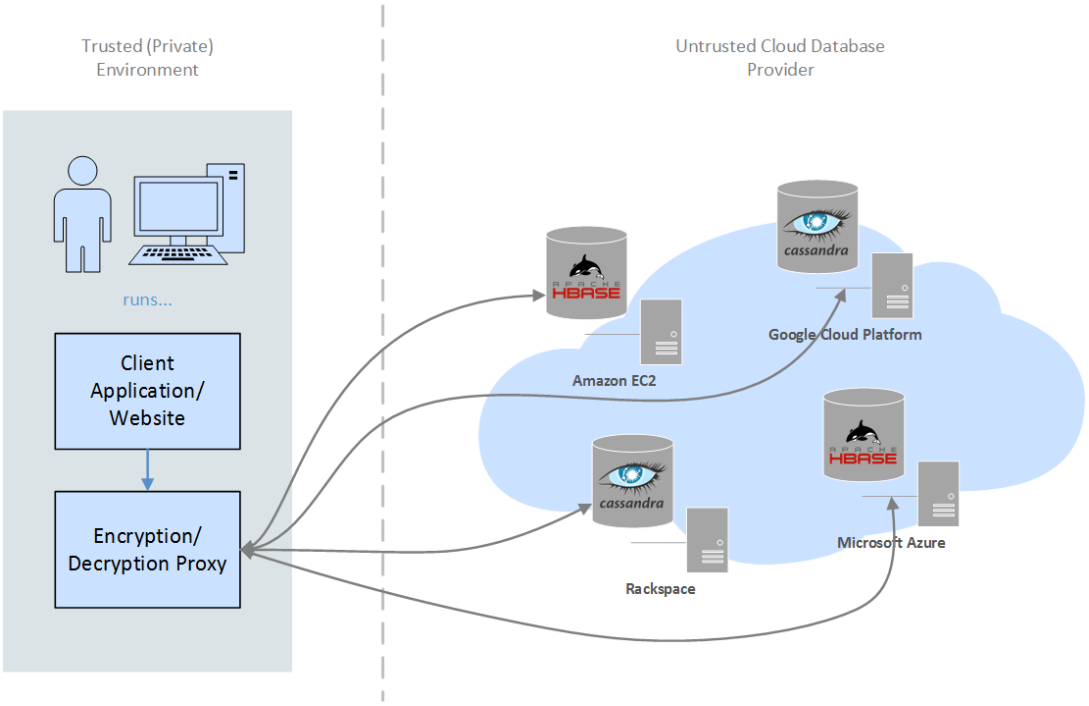


Abb. 2: Gesamtarchitektur des Projekts

3. Kryptographische Verfahren

Es existieren starke kryptographische Verfahren, um den Schutz vertraulicher Daten beweisbar sicherzustellen. Neben dem Vertraulichkeitsschutz gilt es aber auch, die Daten effizient verarbeiten zu können. Der Schutz der Daten und die Nutzbarkeit der Daten sind daher in der Regel zwei gegensätzliche Anforderungen. Idealerweise sollte eine (Vor-)Verarbeitung auf verschlüsselten Daten möglich sein, um eine vollständige Entschlüsselung, Verarbeitung und anschließende Neu-Verschlüsselung zu vermeiden. Um die Funktionalität der Datenbanken nicht zu beeinträchtigen, sollen die Daten je nach Datentyp trotz Verschlüsselung entweder weiterhin durchsuchbar (Text) oder sortierbar (Text, numerische Werte) bleiben. Im Folgenden werden die von uns ausgewählten Verfahren mit Fokus auf deren Eignung kurz vorgestellt.

3.1. Durchsuchbare Verschlüsselung

Abstrakt betrachtet erstellt ein Verfahren zur durchsuchbaren Verschlüsselung einen Suchindex für eine Menge von Daten. Der Suchindex enthält vorher zu definierende Schlüsselworte und die Indexeinträge verweisen dann auf die Datensätze, die das jeweilige Schlüsselwort enthalten. Der Index wird mit einer sog. „Trapdoor“-Funktion verschlüsselt. Eine solche Funktion zeichnet sich dadurch aus, dass sie leicht in eine Richtung zu berechnen ist, aber schwer die Rückrichtung (das Inverse). Ein beliebtes Beispiel hierfür ist die Berechnung des Produktes zweier großer Primzahlen. Dieses Produkt ist leicht auszurechnen. Es ist aber schwierig von diesem Produkt wieder die beiden ursprünglichen Primzahlen abzuleiten, es sei denn, man kennt eine der beiden. Diese wäre dann die sog. Falltür (engl.: trapdoor). Entsprechend ist auf diese Weise analog nur mit Hilfe einer „Trapdoor“ ein Vergleich von einem Suchwort mit den im Index enthaltenen Schlüsselwörtern möglich. Es gibt aber auch Verfahren, die ohne einen solchen Suchindex auskommen. Die Falltür steckt dann direkt in der besonderen Form der Verschlüsselung.

Um die Sicherheit der Verfahren zur durchsuchbaren Verschlüsselung beurteilen zu können, werden verschiedene Einstufungen vorgenommen. Diese geben an, gegen welche Art von Angriff welche Art von Sicherheit erwartet werden kann. Im Wesentlichen können drei Arten von Informationen unerwünscht offenbart werden: Index-Informationen (z.B. Anzahl der Worte pro Dokument, Anzahl der Dokumente, Dokumentlängen, Dokument-IDs etc.), Suchmuster-Informationen (wonach wurde gesucht?) und Zugriffsmuster-Informationen (wie viele Treffer lieferte eine Anfrage im Vergleich zu einer anderen?). Ausführlichere Beschreibungen und Definitionen finden sich in [Curtmola06] und [Bösch14].

Es existiert eine Vielzahl von Schemata zur durchsuchbaren Verschlüsselung [Bösch14]. Eine grobe Kategorisierung kann vorgenommen werden in index-basierte und nicht index-basierte Schemata. Erstere konnten wiederum unterschieden werden nach der Art, in der der Index angelegt wird, entweder mit Einträgen pro durchsuchbarem Dokument oder pro möglichem Suchwort. Wir haben je ein Schema jeder Kategorie implementiert und entsprechende Benchmarks durchgeführt.

(a) Sequential Scan (nicht index-basiert)

Funktionsweise. Wir wählten das Schema von Song et al. [Song00], das in seiner Kategorie alternativlos ist. Sequential-Scan-Verfahren müssen für Verschlüsselung und Suche über den kompletten Datensatz iterieren, was die Anwendbarkeit für sehr große Datenmengen einschränkt. Jedes Wort wird separat verschlüsselt, wobei der Schlüsseltext einen bestimmten Hashwert in einem bestimmten Format enthält. Bei der Suche wird dann wortweise der Hashwert extrahiert und auf diese bestimmte Form überprüft. Dabei werden keine Ressourcen zur Erstellung, Aufrechterhaltung oder Speicherung eines Index benötigt. Das Schema ist somit sehr leicht für ein Datenbank-Szenario umzusetzen. Es eignet sich sowohl für lese- als auch schreibintensive Szenarien gleichermaßen.

Sicherheit. Bei [Song00] kann ein Angreifer nicht zwischen den Geheimtexten zweier gleichlanger Klartexte unterscheiden, selbst wenn er die Klartexte selbst wählen darf.

(b) Index pro Suchwort

Funktionsweise. In dieser Kategorie fiel die Wahl auf das Schema von Curtmola et. al [Curtmola06] aufgrund seiner hohen Effizienz bei der Suche. Der Index besteht hier aus einem Array mit den verschlüsselten Feldern von je einer verketteten Liste pro Suchwort, die randomisiert durchmischt abgelegt werden. Eine zusätzliche verschlüsselte Lookup-Tabelle identifiziert jeweils das erste zu einem Suchwort gehörende Feld. Der Verschlüsselungsvorgang ist also vergleichsweise aufwendig, der Suchvorgang mit einem einzigen Zugriff auf die Lookup Tabelle und dem Auslesen der entsprechenden Felder des Arrays dafür sehr simpel und schnell. Somit eignet sich dieses Schema besonders für Datenbanken, in die wenig geschrieben, aber aus denen oft gelesen wird.

Sicherheit. [Curtmola06] präsentiert zwei Versionen dieses Schemas. In der nicht-adaptiven Version offenbaren Schlüsseltexte und verschlüsselte Indizes keine Informationen, außer der Anzahl der verschlüsselten Dokumente und deren Länge. Eine durchgeführte Suche offenbart keine Informationen, außer den eigentlichen Suchergebnissen. In der adaptiven Version gilt dies auch, wenn der Angreifer seine Angriffsmuster basierend auf bereits durchgeführten Suchen ändern darf.

(c) Index pro Dokument:

Funktionsweise. Implementiert wurde das Schema von Hahn et. al [Hahn14], das auf einem Index mit verschlüsselten Repräsentationen aller Worte pro Dokument basiert, die während des Verschlüsselungsvorgangs erzeugt und gespeichert werden müssen. Zusätzlich speichert ein invertierter Index die Ergebnisse bereits durchgeführter Suchvorgänge, was dazu führt, dass die wiederholte Suche nach einem Wort noch schneller als mit [Curtmola06] durchgeführt werden kann. Dieses Schema eignet sich also besonders für Datenbanken, die oft dieselben Anfragen bearbeiten müssen.

Sicherheit. [Hahn14] ist so sicher wie die adaptive Version von [Curtmola06].

3.2. Ordnungsbewahrende Verschlüsselung

Mit ordnungsbewahrender Verschlüsselung für numerische Daten überträgt sich eine Ordnungsrelation von den Klartexten auf die Schlüsseltexte: wenn für zwei Klartexte a und b gilt, dass a < b, dann gilt für die Verschlüsselungen Enc(a) < Enc(b). Das grundlegende Verfahren wurde in [Agrawal04] entwickelt, [Boldyreva11] liefern aktuelle Erkenntnisse.

Die ordnungsbewahrende Verschlüsselung ist von besonderer Bedeutung für Spaltenfamilien-Datenbanken wie Cassandra und HBase. Sie wird für alle Daten benötigt, die sortierbar bleiben müssen. Die betrifft insbesondere Zeilenidentifikatioren. So werden Zeilen mit ähnlichen Identifikatoren auch physisch nah beieinander abgelegt („Datenlokalität“), was insbesondere Bereichsabfragen extrem beschleunigt. Darüber hinaus sollte ordnungsbewahrende Verschlüsselung bei allen Informationen angewandt werden, die für die Datenbank relevant für Zeitangaben (und somit Versionierungen) sind. Das betrifft in allererster Linie Timestamps, die auch nach der Verschlüsselung noch Auskunft darüber geben können müssen, welche Version eines Datensatzes aktuell ist.

Auch zur ordnungsbewahrenden Verschlüsselung gibt es einige Ansätze, die sich im Wesentlichen aber in zwei Kategorien aufteilen lassen, nämlich ob das Schema zur Ver- und Entschlüsselung Statusinformationen benötigt (zustandsorientiert) oder nicht (nicht zustandsorientiert). Wir untersuchten und implementierten jeweils mindestens ein Schema jeder Kategorie.

(a) Zustandsorientiert

Funktionsweise. Bei dieser Art der ordnungsbewahrenden Verschlüsselung wird ein Zustandsspeicher benötigt (ein Wörterbuch), in dem zu jeder benötigten Zahl aus dem Definitionsbereich (Klartextbereich) ihre Abbildung im Wertebereich (Schlüsseltextbereich) vermerkt wird. Das gestaltet die Entschlüsselung sehr einfach, denn es muss nur der entsprechende Eintrag nachgeschlagen werden. Nach diesem Prinzip arbeiten auch die von uns gewählten Schemata. [Kerschb14] initialisiert das Wörterbuch mit dem Minimal- und Maximalwert; Klartext- und Schlüsseltextbereich sind vorgeben. Zu verschlüsselnde Klartextwerte werden dann nach Bedarf mit einer bestimmten Formel errechnet und an entsprechender Stelle im Schlüsseltextbereich eingefügt. Ist die Reihenfolge der einzufügenden Klartexte ungünstig (z.B. bereits sortiert), kann es passieren, dass an eben dieser Stelle kein Platz mehr ist, d.h. ein neuer Wert zwischen zwei bereits existierenden, direkt aufeinanderfolgenden Schlüsseltextwerten eingefügt werden müsste. In diesem Fall muss eine entsprechende Ausbalancierung des Schlüsseltextbereichs vorgenommen werden, d.h. alle Werte müssen gelesen, neu verschlüsselt und wieder geschrieben werden. Das ist sehr aufwendig, passiert aber vernachlässigbar selten. [Wozniak13] funktioniert ähnlich (mit einem Zufallselement, statt einer Formel), vermeidet dieses Problem aber, indem es den Klartextbereich nicht nach Bedarf, sondern direkt komplett verschlüsselt. Da dies aber nicht sehr praxistauglich ist, haben wir das Verfahren insofern modifiziert, dass es Klartextwerte auch nur nach Bedarf verschlüsselt. Dabei konnten wir erreichen, dass im Hintergrund zwar immer noch deutlich mehr Klartextwerte als eigentlich benötigt verschlüsselt werden müssen. Dies beschränkt sich dann im Gegensatz zum unmodifizierten Algorithmus der Autoren aber auf einen vernachlässigbar kleinen Teil des kompletten Klartextbereiches, so dass eine zu [Kerschb14] vergleichbare Effizienz erreicht werden konnte (siehe Kapitel 4.2), ohne dessen Problem der eventuellen Rebalancierungen und damit Neuverschlüsselungen zu haben. Damit eignet sich [Wozniak13] insbesondere auch zur Verschlüsselung von Zeilenidentifikatoren.

Sicherheit. In [Wozniak13] und [Kerschb14] wird über die Klartexte wird abgesehen von ihren Relationen untereinander keinerlei weitere Information offenbart, jedoch nur, wenn der Wertebereich exponentiell größer ist als der Definitionsbereich.

(b) Nicht zustandsorientiert

Funktionsweise. Wir wählten den Ansatz von [Boldyreva11], der in dieser Kategorie alternativlos ist. Er basiert auf zwei Ideen. Zum einen wird das Wörterbuch dynamisch nur für den Bereich erzeugt, der gerade gebraucht wird, statt es dauerhaft zu speichern. Zum anderen erfolgt die Auswahl einer Zahl aus dem Wertebereich zu einer Zahl aus dem Definitionsbereich anhand der hypergeometrischen Zufallsverteilung. Somit ist zu keinem Zeitpunkt eine Neuordnung/Neuverschlüsselung nötig und es muss dementsprechend kein Zustand gesichert werden. Auch [Boldyreva11] kann somit zur Verschlüsselung von Zeilenidentifikatoren verwendet werden.

Sicherheit. [Boldyreva11] ist so sicher wie [Wozniak13] und [Kerschb14]. Darüber hinaus werden Schlüsseltexte des Wertebereichs gleichmäßig, aber trotzdem zufällig (also „ideal“) den Klartexten des Definitionsbereichs zugeordnet.

4. Umsetzung und Auswertung

Um die Umsetzbarkeit und Performanz verschiedener Verschlüsselungsverfahren in der Realität beurteilen zu können, sind unsere bisher erfolgten Arbeiten geprägt von praktischen Untersuchungen auf den konkreten Datenbanken Apache Cassandra und Apache HBase, da derartige Informationen in den verwandten Arbeiten mit ihrem zumeist theorielastigen Charakter nicht existieren. Es wurde eine eigene Umgebung für Benchmarks entworfen, die im Wesentlichen zwei Ziele umsetzt. Zum einen lassen sich vorgestellten Schemata für durchsuchbare und ordnungsbewahrende Verschlüsselungen direkt integrieren. Zum anderen ist es möglich, beliebige Testdatensätze zu benutzen. Dies ermöglichte nun realitätsnahe Analysen und die Nutzung von standardisierten Datensätzen.

Alle Implementationen[3] erfolgten komplett in Java 8. Zur Umsetzung der Verschlüsselungsalgorithmen kamen sowohl die mitgelieferte JCE (Java Cryptography Extension) zum Einsatz, als auch externe Bibliotheken, insbesondere die Bouncy Castle Crypto API[4]. Der Verschlüsselungsschritt ist jeweils dem Datenbankzugriff vorgelagert und der Entschlüsselungsschritt wird auf den Antworten des Datenbankservers ausgeführt. Die Datenbanken selbst wurden wie angestrebt nicht modifiziert. Die Ergebnisse wurden dabei auf Standardnotebook-Hardware (Intel Core i7-4600 CPU bei 2,1 GHz, 8 GB RAM) erzielt. Mit den Ressourcen verteilter Cloud Computing Plattformen sind entsprechend deutliche Performanzsteigerungen zu erwarten. Wir planen derartige Tests für die nahe Zukunft.

4.1. Durchsuchbare Verschlüsselung

Ausführliche Ergebnisse für die drei Verfahren zur durchsuchbaren Verschlüsselung finden sich in unseren bereits veröffentlichten Vorarbeiten [Waage15a, Waage15b], exemplarisch an dieser Stelle eine Evaluation, für die wir die ersten 10.000 Mails des TREC Public Spam Corpus [Cormack05] verschlüsselten, die zusammen ca. 7.000.000 Worte enthalten. Anschließend wurden darin einzelne, zufällig ausgewählte Worte gesucht.

[3] Verfügbar unter https://github.com/dbsec/FamilyGuard
[4] www.bouncycastle.org, besucht am 24.03.2016

Abbildung 2 zeigt die erzielten Resultate unter dem Einsatz von Apache Cassandra. Mit Apache HBase erzielten wir vergleichbare Ergebnisse. Dargestellt ist jeweils der Mittelwert aus mindestens drei Messungen. Unsere Implementation von [Song00] zeigt, dass trotz des iterativen Ansatzes über 500.000 Worte pro Sekunde durchsucht und etwa 160.000 Worte pro Sekunde verschlüsselt werden können. [Hahn14] ist minimal schneller mit ca. 550.000 Worten pro Sekunde in der Suche, dafür mit 120.000 Worten pro Sekunde beim Verschlüsseln etwas langsamer. Ein anderes Bild ergibt sich beim Ansatz von [Curtmola06]. Hier macht sich die sehr aufwendige Verschlüsselung bemerkbar, so dass unsere Implementierung nur 23.000 Worte pro Sekunde [Waage15] erreicht. Die Suche ist dafür optimal schnell, da in der Lookup-Tabelle nur das entsprechende Wort nachgeschlagen werden muss. Selbst wenn der Datensatz viele Millionen Worte umfasst, werden dafür nur wenige Millisekunden benötigt.

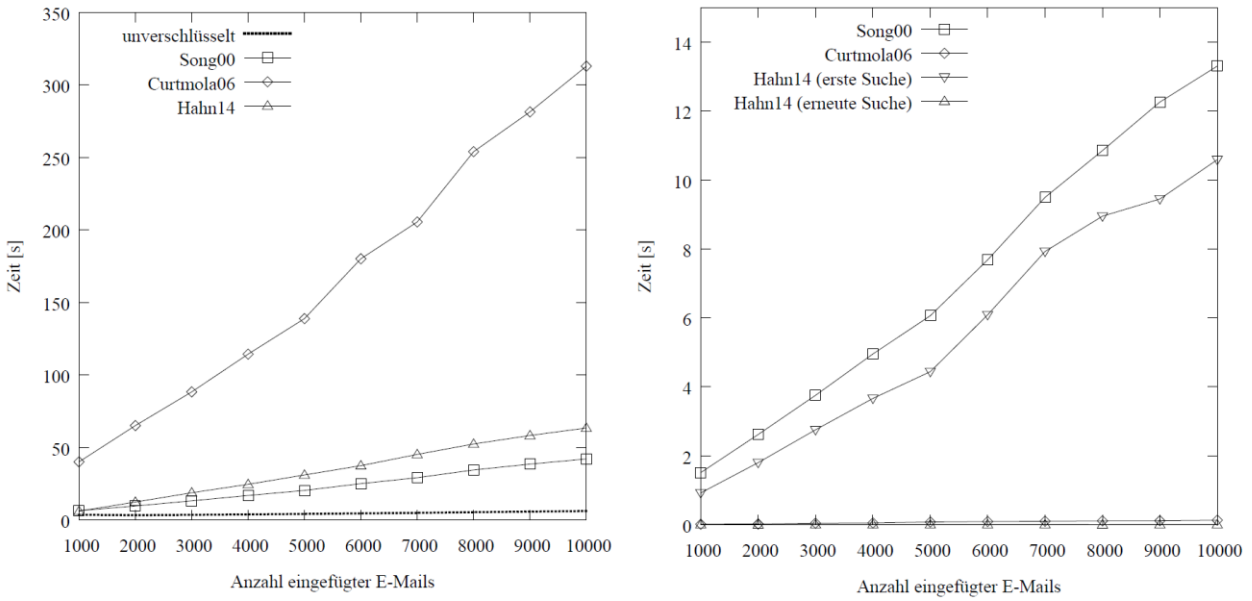


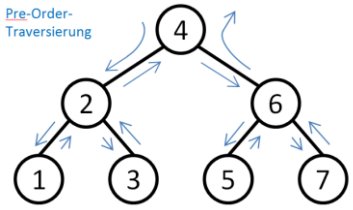
Abb. 3: Zur Verschlüsselung (links) und Suche (rechts) benötigte Zeit mit steigender Datensatzgröße

Obwohl die absoluten Zahlen intuitiv auf eine praxistaugliche Performanz schließen lassen, ist der Preis für die Verschlüsselung verglichen zum unverschlüsselten Betrieb relativ hoch, beginnend bei ca. 50% Geschwindigkeitseinbußen für [Song00] und sogar 90% für [Curtmola06]. Da die Datenbank für größeren Input zunehmend effizienter wird, verschlechtert sich diese Bilanz darüber hinaus je größer der Datensatz wird. Demgegenüber steht allerdings auch der Zugewinn einer neuen Funktionalität, die Spaltenfamiliendatenbanken sonst gar nicht hätte: die Suche nach einzelnen Worten innerhalb einer Tabellenzelle. Out-of-the-box kann nämlich abgesehen von den Zeilenidentifikatoren gar nichts durchsucht werden. Für eine Suche in anderen Spalten muss vorher explizit ein sekundärer Index angelegt werden und selbst damit kann dann nur nach dem kompletten Inhalt einer Zelle gesucht werden, nicht aber nach einzelnen Worten (daher „fehlt“ die entsprechende Kurve auch in Abbilung 2, rechts).

Die sehr unterschiedlichen Ergebnisse der einzelnen Schemata legen darüber hinaus nahe, dass es bereits bei deren Auswahl vorteilhaft ist, zu wissen, wie die Datenbank später hauptsächlich benutzt werden soll. Soll überwiegend geschrieben werden, empfiehlt sich [Song00], da es die Verschlüsselung am schnellsten abwickelt. Für sogenannte „write-once“ Datenbanken, in die selten geschrieben, die aber oft angefragt werden, empfiehlt sich [Curtmola06]. [Hahn14] ist ein Kompromiss aus beiden. Hier ist die Verschlüsselung vergleichsweise flott und insbesondere wiederkehrende Anfragen können sehr schnell beantwortet werden.

4.2. Ordnungsbewahrende Verschlüsselung

Hier verschlüsseln wir für eine Evaluation der Performance mit Hilfe der drei im Kapitel 3.2.2. vorgestellten Verfahren bis zu 20.000 zufällig ausgewählte, im Klartextbereich normalverteilte Zahlen und führen die notwendigen Schreib- und Leseoperationen mit Cassandra und HBase durch. Für einen sicheren Einsatz von ordnungsbewahrender Verschlüsselung sollte die Schlüsseltextlänge möglichst deutlich größer sein als die Klartextlänge. Für einen praxisrelevanten Test wählen wir daher für den Klartext eine Größe von 32 Bit (was dem gewöhnlichen „Integer“ Datentyp entspricht) und für den Schlüsseltext 64 Bit (entsprechend dem „Long“ Datentyp). Während die Einfügereihenfolge der Zahlen für [Boldyreva11] und [Wozniak13] egal ist, müssen für [Kerschb14] drei Fälle unterschieden werden. Im besten Fall werden alle Elemente eines perfekt balancierten binären Suchbaums in Pre-Order Traversierung eingefügt (vgl. Abbildung 4), im durchschnittlichen Fall werden die Elemente einfach in zufälliger Reihenfolge eingefügt und im schlechtesten Fall sind die Elemente vorsortiert.



Pre-Order Traversierung	4	2	1	3	6	5	7
zufällig	3	5	2	6	7	4	1
sortiert	1	2	3	4	5	6	7

Abb. 4: Einfaches Beispiel für das Einfügen der Zahlen 1 bis 7 für die drei zu unterscheidenden Fälle des [Kerschb14] Algorithmus

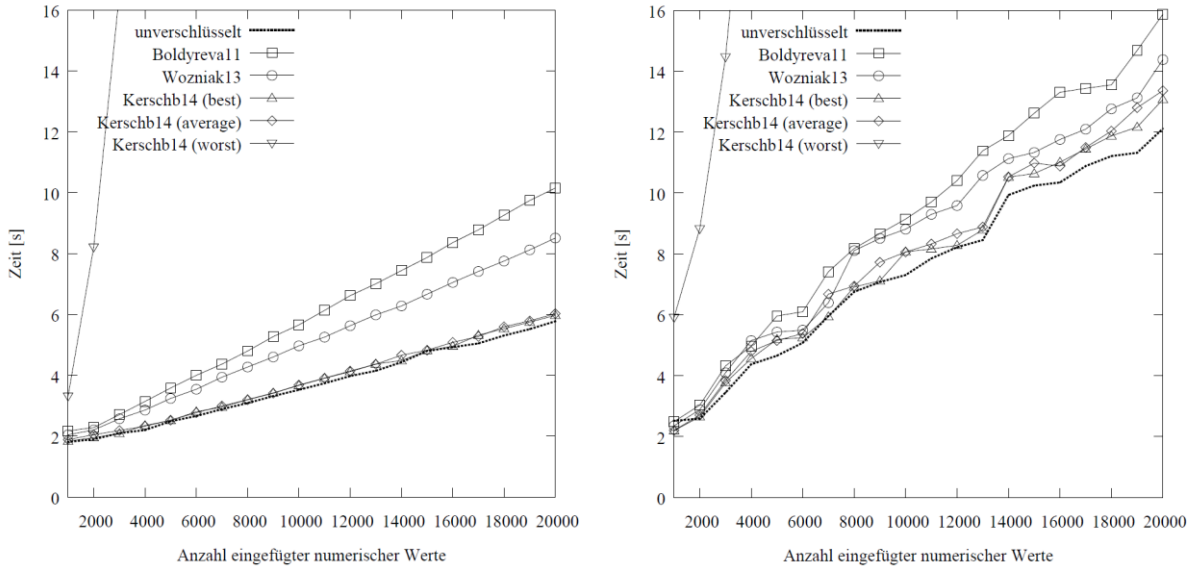


Abb. 5: Dauer der Verschlüsselung mit Cassandra (links) und HBase (rechts)

Abbildung 4 zeigt die Resultate. Dargestellt ist jeweils der Mittelwert aus mindestens zehn Messungen. Cassandra ist verglichen zu HBase im Schnitt etwa um ein Drittel schneller. Die Erklärung hierfür liegt darin, dass [Kerschb14] und [Wozniak13] so schnell sind, dass die Zeit der reinen Einfügeoperation in die Datenbank einen signifikanten Teil der Gesamtzeit der Verschlüsselung ausmacht. Da Cassandra optimiert ist auf Schreibvorgänge, zeigt sich hier der entsprechende Vorteil. Eine Ausnahme bildet der schlechteste Fall von [Kerschb14], bei dem neben den Schreibvorgängen auch oft aus der Datenbank gelesen werden muss. Dies schlägt sich nieder in einem ca. 12-15%igen Vorsprung für HBase, das im Gegensatz zu Cassandra auf Lesevorgänge optimiert ist.

Da die Entschlüsselung sehr simpel ist, verzichten wir auf entsprechende Abbildungen. Im Fall von [Wozniak13] und [Kerschb14] besteht sie aus einem Nachschlagevorgang im entsprechenden Wörterbuch. Dies dauert auch bei großen Datensätzen in der Regel weniger als 1 ms. Das nicht zustandsbasierte Verfahren von [Boldyreva11] kann auf einen solchen Index nicht zurückgreifen. Die notwendigen Berechnungen dauerten unabhängig von der Datensatzgröße im Schnitt 5 ms.

Auffällig ist der verglichen zur durchsuchbaren Verschlüsselung geringere Geschwindigkeitsverlust. Im besten und durchschnittlichen Fall von [Kerschb14] beträgt gerade einmal 2-4%. Wenn man den schlechtesten Fall von [Kerschb14] ausklammert (d.h. wenn man es vermeidet [Kerschb14] mit vorsortiertem Input zu benutzen) verliert man maximal 57% Geschwindigkeit. Es macht also auch hier Sinn, sich je nach Anforderung für ein bestimmtes Schema zu entscheiden. Möchte man in jedem Fall den zusätzlichen Speicherplatzbedarf für die Sicherung eines Wörterbuchs vermeiden, so kommt nur [Boldyreva11] in Frage. Ist nicht zu befürchten, dass der Input bereits stark vorsortiert ist und man Wert auf Geschwindigkeit legt, sollte man auf [Kerschb14] zurückgreifen. [Wozniak13] ist ein Kompromiss aus beiden. Es benötigt zwar ein Wörterbuch, ist aber trotzdem schnell und unabhängig vom Input.

5. Verwandte Arbeiten

Obwohl die Sicherheit von Clouddatenbanken in letzter Zeit mehr und mehr Aufmerksamkeit erfährt, gibt es kaum umfassende Lösungen. Als einzige quell-offenen Implementierung bietet das Projekt CryptDB [Popa12] zumindest für SQL-Datenbanken eine sogenannte SQL-bewusste Verschlüsselung (engl.: „SQL-aware encryption“). Nach dem Vorbild einer Zwiebelschale verschlüsselt CryptDB einen Wert mehrfach (zum Beispiel deterministisch oder ordnungsbewahrend innerhalb einer starken Verschlüsselung). Je nach konkreter Anfrage wird dann eine Lage der Verschlüsselung (engl.: „onion encryption layer“) entfernt, bis das Datenbanksystem die Anfrage auf den verschlüsselten Daten ausführen kann. Jedoch ist der CryptDB-Prototyp nur unzureichend dokumentiert und unterstützt nur traditionelle SQL-Datenbanken (und auch dies nur mit Einschränkungen). Daher ist er nicht für NoSQL-Datenbanken erweiterbar. Zudem bietet der Prototyp keine benutzerfreundliche Schnittstelle und ist daher für den breiten Einsatz ungeeignet.

Weitere Ansätze verlangen teure kryptographische Hardware auf Seiten der Clouddatenbank (wie etwa TrustedDB [Bajaj14] oder Cipherbase [Arasu13]). Es findet dann eine Entschlüsselung der Daten auf Seiten der Clouddatenbanken innerhalb der kryptographischen Hardware statt und der Cloudanbieter benötigt dazu immer den Entschlüsselungsschlüssel des Benutzers; daher sind diese Hardware-basierten Ansätze für einen selbstbestimmten Datenschutz nicht geeignet.

6. Zusammenfassung

Unsere bisherigen Arbeiten setzen kryptographische Sicherheitsfunktionen für Spaltenfamilien-Datenbanken um und analysieren sie eingehend – idealerweise ohne die generelle Funktionsweise (und damit die bestehenden Implementierungen) von Spaltenfamilien-Datenbanken ändern zu müssen.

Die bisher erfolgten Arbeiten zeigen, dass die untersuchten Datenbanken sinnvoll mit verschlüsselten Daten benutzt werden können. Die Geschwindigkeitseinbußen bewegen sich in einem akzeptablen Rahmen und werden durch die erzielten Sicherheitseigenschaften sehr gut ausgeglichen.

Aus den existierenden Verfahren zur durchsuchbaren und ordnungsbewahrenden Verschlüsselung konnten solche Schemata identifiziert und implementiert werden, die den Anforderungen an den praktischen Einsatz in den unmodifizierten Datenbanksystemen ermöglichen und die jeweils bestmöglichen Sicherheitseigenschaften in ihrer

Kategorie aufweisen. Die Performanz konnte mit praxisorientierten Messungen quantifiziert werden. Die bisherigen Ergebnisse legen nahe, dass der verschlüsselte Datenbankbetrieb praktikabel und ausreichend schnell ist.

Danksagung

Das Projekt FamilyGuard wird von der DFG gefördert. Förderkennzeichen: WI 4086/2-1.

Referenzen

[Arasu13] Arasu, A., Blanas, S., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R., & Venkatesan, R. (2013, January). Orthogonal Security with Cipherbase. In CIDR.

[Bajaj14] Bajaj, S., & Sion, R. (2014). TrustedDB: A trusted hardware-based database with privacy and data confidentiality. Knowledge and Data Engineering, IEEE Transactions on, 26(3), 752-765.

[Boldyreva09] Boldyreva, A., Chenette, N., Lee, Y., & O'neill, A. (2009). Order-preserving symmetric encryption. In Advances in Cryptology-EUROCRYPT 2009 (pp. 224-241). Springer Berlin Heidelberg.

[Boldyreva11] Boldyreva, A., Chenette, N., & O'Neill, A. (2011). Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Advances in Cryptology–CRYPTO 2011 (pp. 578-595). Springer Berlin Heidelberg.

[Borthakur11] Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., ... & Aiyer, A. (2011, June). Apache Hadoop goes realtime at Facebook. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 1071-1080). ACM.

[Bösch14] Bösch, C., Hartel, P., Jonker, W., & Peter, A. (2014). A survey of provably secure searchable encryption. ACM Computing Surveys (CSUR), 47(2), 18.

[Brewer10] Brewer, E. (2010, July). A certain freedom: thoughts on the cap theorem. In Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing (pp. 335-335). ACM.

[Byun12] Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Kepner, J., ... & Prout, A. (2012, September). Driving big data with big compute. In High Performance Extreme Computing (HPEC), 2012 IEEE Conference on (pp. 1-6). IEEE.

[Cormack05] Cormack, G. V., Lynam, T.R. (2005). TREC Spam Track Overview.

[Curtmola06] Curtmola, R., Garay, J., Kamara, S., & Ostrovsky, R. (2006, October). Searchable symmetric encryption: improved definitions and efficient constructions. In Proceedings of the 13th ACM conference on Computer and communications security (pp. 79-88). ACM.

[Hahn14] Hahn, F., & Kerschbaum, F. (2014, November). Searchable Encryption with Secure and Efficient Updates. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (pp. 310-320). ACM.

[Kerschb14] Kerschbaum, F., & Schroepfer, A. (2014, November). Optimal Average-Complexity Ideal-Security Order-Preserving Encryption. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (pp. 275-286). ACM.

[Lakshman10] Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, *44*(2), 35-40.

[Popa12] Popa, R. A., Redfield, C., Zeldovich, N., & Balakrishnan, H. (2012). CryptDB: Processing queries on an encrypted database. Communications of the ACM, 55(9), 103-111.

[Song00] Song, D. X., Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. In Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on (pp. 44-55). IEEE.

[Waage15] Waage, T., Jhajj, R. S., Wiese, L. (2015). Searchable Encryption in Apache Cassandra. In Foundations and Practice of Security - 8th International Symposium (FPS2015), Lecture Notes in Computer Science, 9482. Springer.

[Wiese15] Wiese L. (2015). Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases, De Gruyter Verlag.

[Wozniak13] Wozniak, S., Rossberg, M., Grau, S., Alshawish, A., & Schaefer, G. (2013, November). Beyond the ideal object: towards disclosure-resilient order-preserving encryption schemes. In Proceedings of the 2013 ACM workshop on Cloud computing security workshop (pp. 89-100). ACM.