Ordnungserhaltende Verschlüsselung in Cloud-Datenbanken

Tim Waage · Lena Wiese

Georg-August-Universität Göttingen
Institut für Informatik
Forschungsgruppe Knowledge Engineering
{tim.waage | lena.wiese}@cs.uni-goettingen.de

Zusammenfassung

Ordnungserhaltende Verschlüsselung (engl. order-preserving encryption, kurz: OPE) liefert Schlüsseltexte, die die relative Ordnung der darunterliegenden Klartexte bewahren. Damit ist es gut geeignet für Bereichsabfragen (engl. range queries) über verschlüsselte Daten und somit ein populärer Anwendungsfall für ausgelagerte Datenbanken, insbesondere für die sog. Cloud- Datenbanken. Leider haben viele OPE-Schemata praktische Nachteile, wie beispielsweise die Notwendigkeit komplizierter Datenstrukturen oder zusätzlicher Soft-/Hardware-Komponenten. Während es für OPE in der Theorie also viele Ansätze gibt, finden sich daher kaum Implementationen in der Praxis. Die vorliegende Arbeit befasst sich daher mit der Frage nach den praktischen Anforderungen für den Einsatz von OPE in realitätsnahen Szenarien mit besonderem Fokus auf Cloud-Datenbanken. Wir evaluieren eine Reihe verschiedener OPE-Verfahren in Bezug auf Praxistauglichkeit, schlagen für zwei von ihnen ([WRGA+13, KeSc14]) Verbesserungen vor, die wir anschließend auch implementieren. Ein konkreter Benchmark liefert dann Laufzeitanalysen und einen Vergleich zur Weiterentwicklung ([BoCO11]) des einzigen uns bekannten OPE-Verfahrens ([BCLO09]), das praktisch verwendet wird (beispielsweise in [PRZB11]). Als Grundlage dafür nutzen wir zwei populäre NoSQL Spaltenfamiliendatenbanken und zeigen damit die praktischen Stärken und Schwächen der verschiedenen OPE-Verfahren.

1 Einleitung

In sogenannten Big Data Anwendungen werden große Datenmengen von Datenbanken aufgenommen und verarbeitet. Insbesondere moderne Web-Services haben hohe Anforderungen an Verfügbarkeit, Konsistenz, Performance und Skalierbarkeit, die mit traditionellen relationalen Datenbanken nur schwer realisiert werden können. NoSQL (Not only SQL) Datenbanken, vor allem deren Subkategorie der Spaltenfamiliendatenbanken (im folgenden kurz: SFD), wurden vor diesem Hintergrund konzipiert. Sie laufen in verteilten Umgebungen und sind die Schlüsseltechnologie hinter vielen populären Plattformen, wie zum Beispiel Apache HBase hinter Facebook [BGSM+11], Apache Cassandra hinter eBay or Googles BigTable hinter fast jedem der über 60 Google Services [CDGH+08]. Aufgrund des täglich steigenden Datenaufkommens, sei es in Social Media Netzwerken, durch Geschäftsprozesse oder in der Forschung, werden diese Datenbanken häufig auf Server ausgelagert, die nicht unter der eigenen Kontrolle stehen und somit nicht vertrauenswürdig für sensible Daten sind. Leider bieten NoSQL Datenbanken in der Regel aber kaum Sicherheitsmechanismen [OGOGG+11].

Verschlüsselung ist grundsätzlich eine wirksame Maßnahme zum Schutz vertraulicher Daten

P. Schartner (Hrsg.) · D•A•CH Security 2016 · syssec (2016) 75-86.

in solchen Umgebungen. Sie limitiert aber auch die Möglichkeiten zur Datenverarbeitung. Der Einsatz traditioneller Techniken wie RSA oder AES ist für die Praxis untauglich, denn er bewahrt nicht die Eigenschaften der Klartexte, die eine SFD zur Beantwortung von Anfragen braucht. Eine oft genutzte Form solcher Anfragen sind Bereichsabfragen, die über Datenreihen in kontinuierlich sortierten Blöcken ausgeführt werden. Um eine entsprechende Sortierung auch im verschlüsselten Datensatz zu gewährleisten, muss in diesem die Ordnungsrelation der Klartexte weiter erhalten werden.

Obwohl ordnungserhaltene Verschlüsselung ein sehr aktives Forschungsfeld ist, so ist die Praxistauglichkeit der entsprechenden Ansätze oftmals nicht ausreichend. Die vorliegende Arbeit leistet daher folgendes:

- Die grundlegenden Anforderungen an OPE-Verfahren für den Einsatz in SFD werden identifiziert.
- Anhand dieser Anforderungen wird eine Reihe existierender OPE-Verfahren im Hinblick auf Praxistauglichkeit evaluiert. Für zwei dieser Verfahren werden Modifikationen vorgeschlagen, die deren Praxistauglichkeit verbessern.
- Anschließend werden die beiden vorgeschlagenen Modifikationen und ein weiteres OPE-Verfahren implementiert und in Laufzeitanalysen mit den beiden zur Zeit populärsten[1] SFD verglichen, namentlich Apache Cassandra [LaMa10] und Apache HBase [BGSM+11]. Dabei werden Stärken und Schwächen der Verfahren und Datenbanken aufgezeigt.

2 Hintergrund

2.1 Ordnungserhaltende Verschlüsselung

In Datenbank-Anwendungen erlaubt OPE Vergleiche und Sortierungen wie mit Klartextdaten. Das Erstellen von Indizes und Bereichsabfragen funktionieren also noch genauso, obwohl die Daten verschlüsselt sind. Formal definiert erfüllt ein Schema zur (symmetrischen) ordnungserhaltenden Verschlüsselung von einem Klartextbereich D (Domain) in einen Schlüsseltextbereich R (Range) mit den Algorithmen $(KGen, Enc, Dec)$ die folgenden Bedingungen:

- Der Algorighmus $KGen$ generiert einen zufälligen Schlüssel k.
- Der Verschlüsselungsalgorithmus Enc generiert aus k und dem Klartext p den Schlüsseltext $c = Enc_k(p)$.
- Der Entschlüsselungsalgorithmus Dec generiert aus k und einem Schlüsseltext c den Klartext p.
- es gilt: $Dec_k(Enc_k(p)) = p$
- $p_1 \leq p_2 \Rightarrow Enc_k(p_1) \leq Enc_k(p_2)$ für alle $p_1, p_2 \in D$

[AKSX04] lieferte die erste formale OPE Definition und adressierte sie mit einem theoretischen Schema.

[1] Solit-IT: DB-engines ranking - `http://db- engines.com/en/ranking` (alle URLs wurden geprüft am 16.03.2016)

2.2 Spaltenfamilien-Datenbanken

Verschiedene SFD folgen verschiedenen Prinzipien in Bezug auf Architektur, Anfragesprachen, Datentypen, etc. Eine hervorragende Übersicht bietet [Harr15]. Trotz dieser Heterogenität nutzen aber alle SFD ein sehr ähnliches Datenmodel, das wie folgt beschrieben werden kann.

Während SFD Tabellen, Zeilen und Spalten ähnlich wie traditionelle (SQL-basierte) Datenbanken verwenden, so besteht der wesentliche Unterschied jedoch darin, dass Spalten für jede Zeile individuell erstellt und nicht durch die Tabellenstruktur vorgegeben werden. Eine Zeile muss insbesondere nicht für jede Spalte einen Wert enthalten. Zwei unterschiedliche Zeilen können vollkommen disjunkte Spaltenmengen haben. Jede Zeile hat jedoch einen Identifikator (engl. row key), der in der Tabelle einzigartig sein muss. Die Datenbank hält alle Zeilen permanent in der nach dem Zeilen-Identifikator *sortierten* Reihenfolge vor und verteilt sie so über die Server eines Clusters, dass aufeinanderfolgende Zeilen möglichst auf dem gleichen Server liegen (z.B. Reihe 1-10 auf Server 1, Reihe 11-20 auf Server 2, etc.). Bereichsabfragen benötigen so immer nur die Kommunikation zu einer minimalen Anzahl an Knoten. Die kleinsten Informationseinheiten sind Schlüssel-Wert-Paare, bei denen sich der Schlüssel jedoch aus mehreren Komponenten zusammen setzt. Der Inhalt jedes Wertes kann über seinen Schlüssel ermittelt werden.

Zusammenfassend besteht das Datenmodel von SFD also aus multidimensionalen (verteilten) Abbildungen der Form $(Tabelle, Zeilen\text{-}Identifikator, Spalte, Zeitstempel) \rightarrow Wert$, detailliert zum Beispiel beschrieben in [CDGH$^+$08].

3 Praktikabilität von OPE in SFD

3.1 Kriterien

Aufgrund der in Abschnitt 2.2 beschriebenen Gemeinsamkeiten im Datenmodel der SFD müssen OPE-Schemata in der Praxis bestimmte Anforderungen erfüllen, auf die deren Autoren wenn überhaupt oft nur theoretisch eingehen. Wir evaluieren die Praxistauglichkeit von OPE im SFD-Szenario anhand der folgenden fünf Kriterien:

(I) Schlüsseltext-Veränderlichkeit: Wir nennen den von einem OPE Schema generierten Schlüsseltext *veränderlich*, wenn er im fortschreitenden Prozess der Verschlüsselung eines Datensatzes neu verschlüsselt werden muss. Ein Beispiel dafür findet sich im Algorithmus von [KeSc14]. Dessen Zustand ist eine Menge Klartext-Schlüsseltext-Paare, initialisiert mit $\{(-1,-1),(maxKlartextWert, maxSchlüsseltextWert)\}$. Ein neuer Schlüsseltext-Wert wird immer in der Mitte zwischen dem nächst kleineren und dem nächst größeren bereits verschlüsselten Wert eingesetzt. Sind diese beiden Werte aber bereits direkt aufeinanderfolgend, so ist hier kein Platz mehr, um einen neuen Wert aufzunehmen und alle bereits verschlüsselten Werte müssen neu verschlüsselt werden, um sie wieder gleichmäßig im Schlüsseltextbereich zu verteilen. Das bedeutet in der Praxis das Auslesen und wieder Zurückschreiben aller bereits verschlüsselten Werte in die Datenbank. Im Gegensatz dazu nennen wir den von einem OPE-Schema generierten Schlüsseltext *unveränderlich*, wenn er final ist und im Zuge der weiteren Verschlüsselung des Datensatzes nicht mehr geändert werden muss. Ein Beispiel hierfür findet sich im Algorithmus von [WRGA$^+$13], in dessen Verschlüsselung sichergestellt wird, dass zwischen zwei Schlüsseltexten immer noch genügend Abstand gehalten wird, um theoretisch alle dazwischenliegenden Klartextverschlüsselungen aufzunehmen.

Wie in Abschnitt 2.2 bereits erläutert, sortieren SFD die Zeilen einer Tabelle stets nach ihrem Identifikator. Damit dies auch nach der Verschlüsselung weiterhin möglich ist, ist die Anwendung einer ordnungserhaltenden Verschlüsselung an dieser Stelle essentiell. Hierfür kommen darüber hinaus nur OPE-Schemata mit unveränderlichen Schlüsseltexten in Frage, da ansonsten die Gefahr bestünde, dass sich die Zeilen-Identifikatoren ändern. Da diese aber maßgeblich beeinflussen, wie die SFD die Tabelle im Cluster verteilt, könnte eine Änderung des Zeilen-Identifikators auch eine entsprechende Umschichtung der Daten zur Folge haben, um das Datenmodell der SFD in einem konsistenten Zustand zu halten. Das wäre sehr aufwendig und wird daher von SFD auch nicht unterstützt. OPE-Verfahren mit veränderlichen Schlüsseltexten können im Gegensatz dazu aber für andere Spalten sehr wohl benutzt werden, um eine bessere Performance zu erzielen (vgl. Abschnitt 4). Die Schlüsseltext-Veränderlichkeit steht häufig in engem Zusammenhang mit den Kriterien II und V.

(II) Notwendigkeit für zusätzliche Datenstrukturen: Wenn sie nicht zustandslos sind, benötigen OPE-Verfahren zusätzliche Datenstrukturen, um ihren Zustand zu sichern. Dieser besteht in der Regel mindestens aus einer Reihe von Klartext-Schlüsseltext-Paaren, organisiert in Indizes (Wörterbücher, Baumstrukturen, etc.) auf Client-Seite (z.B. [KeSc14]) oder zumindest in einer vertrauenswürdigen Umgebung (z.B. [RACY15]). Insbesondere Baumstrukturen sind für SFD aber relativ aufwendig zu verwalten. Daher bedienen sich manche OPE-Verfahren auch zusätzlicher serverseitiger Komponenten (siehe Kriterium III).

(III) Notwendigkeit zusätzlicher architektonischer Komponenten: Client-Anwendungen und Datenbanken besitzen keine native Unterstützung für OPE-Verfahren. Daher müssen zusätzliche Komponenten eingeführt werden, die Klartext-Anfragen umformulieren, um mit den verschlüsselten Datenstrukturen zu funktionieren und auch die eigentliche Ver- und Entschlüsselung zu übernehmen. Üblicherweise werden diese Komponenten in einer vertrauenswürdigen Umgebung platziert. Es gibt jedoch auch OPE-Verfahren, die zusätzlich serverseitige Komponenten vorsehen (z.B. [PoLZ13]), was einen höheren praktischen Aufwand zur Folge hat und insbesondere inkompatibel zu der überwiegenden Mehrzahl der Database-as-a-Service Angebote ist.

(IV) Vielseitigkeit der Eingabe: Die Autoren aller von uns untersuchten OPE-Verfahren gehen stets von einer positiven ganzzahligen Eingabe aus. Das deckt sich aber nicht mit der Realität, in der man auch negative Zahlen und Gleitkommawerte in Datenbanken speichern möchte. Negative Zahlen könnte man mit der Addition eines ausreichend hohen Offsets vor Verschlüsselung bzw. nach Entschlüsselung beseitigen, allerdings bestünde dabei das Problem, dass man die Größe dieses Offsets nicht bestimmen kann, ohne vorher den kompletten Datensatz zu kennen, was in der Realität oftmals nicht der Fall ist. Überhaupt haben viele OPE-Verfahren den praktischen Nachteil, vor Beginn der Verschlüsselung die zu verschlüsselnden Daten detailliert kennen zu müssen (z.B. [LiWa12]). Der Umgang mit Gleitkommazahlen ist ebenso kompliziert, denn diese lassen sich nicht mit beliebiger Präzision in Ganzzahlen überführen. Es stellt sich also die Frage, inwiefern existierende OPE-Verfahren auch mit negativen Werten und Gleitkommazahlen betrieben werden können. Wir untersuchen diesen Aspekt in Abschnitt 3.2.

Ein anderes praktisches Problem vieler OPE-Verfahren ist die Tatsache, dass sie nur den kompletten Klartextbereich verschlüsseln können, nicht aber gezielt nur bestimmte Werte nach Bedarf (z.B. [WRGA+13, LCYJ+14]). Für den durchaus gebräuchlichen Fall, dass D in der Praxis von einem 32-Bit Integer-Wert vorgegeben wird, wären also $2^{32} = 4294967296$ Verschlüsselungen notwendig, selbst wenn nur wenige davon tatsächlich benötigt würden.

(V) Sicherheit: Die erste formelle Untersuchung von Sicherheit in OPE-Verfahren findet sich in [BCLO09]. Die Autoren beweisen hier, dass ideale Sicherheit ("IND-OCPA", *indistinguishability under an ordered chosen plaintext attack*, d.h. Schlüsseltexte offenbaren nichts als ihre relative Ordnung untereinander) mit unveränderlichen Schlüsseltexten nur erreicht werden kann, wenn R exponentiell größer ist als D, was in der Praxis für große Zahlen schwierig zu erreichen ist. Zum Ausgleich bieten verschiedene OPE-Verfahren verschiedene Ansätze (was sich häufig direkt auf die Kriterien II und III auswirkt). Beispiele sind ein modularer Versatz der Klartexte in [BoCO11] (leicht zu implementieren, aber nur ein kleiner Sicherheitsgewinn) oder die Verschleierung der Anfrageverteilung mittels Fake-Anfragen in [MCOK+15] (was einen erheblichen zusätzlichen Kommunikations- und Rechenaufwand bedeutet). In der Praxis erzielen OPE-Verfahren mit veränderlichen Schlüsseltexten meist die besseren Sicherheitseigenschaften, da hier die Größenanforderung an R nicht besteht. Darüber hinaus können sie aufgrund der in Kriterium I beschriebenen Neuverschlüsselungen die Verteilung von Klartexten in D besser verbergen. Oftmals erreichen sie sogar fast eine Gleichverteilung, die kaum Rückschlüsse zulässt (z.B. in [WRGA+13]). Da diese Neuverschlüsselungen aber aufwendig sind, versuchen OPE-Verfahren zumeist die Anzahl deren Auftretens so gering wie möglich zu halten (z.B. in [KeSc14]) oder sie auf die Serverseite zu verlagern (z.B. in [PoLZ13]), um zumindest den dafür notwendigen zusätzlichen Kommunikationsaufwand zu reduzieren. Eine Alternative zur totalen Vermeidung von Neuverschlüsselungen ist die Behandlung das gesamten Klartextbereichs D, wie in Kriterium IV beschrieben.

3.2 Auswahl geeigneter OPE-Verfahren

Abschnitt 3.1 zeigt, dass OPE-Verfahren mit bestimmten positiven Eigenschaften dafür meist an anderer Stelle Nachteile haben. Eine Übersicht bietet Tabelle 1, welche die von uns untersuchten Schemata bzgl. der fünf eingeführten Kriterien evaluiert. Für unsere Implementation und die praktischen Tests mit den SFD wählten wir die drei vielversprechendsten Verfahren aus und beschreiben deren Ideen und unsere Modifikationen in den Abschnitten 3.2.1 bis 3.2.3.

Tab. 1: Stärken und Schwächen verschiedener OPE-Verfahren

OPE-Verfahren	I	II	III	IV	V
Kadhem et. al, '10	+	−−	+	−	+
Boldyreva et. al, '11	+	++	+	−	+
Liu & Wang, '12	+	−−	+	−−	?[2]
Popa et. al, '13	−	−−	−	++	++
Wozniak at. al, '13	+	−	+	+	++
Liu et. all, '14	+	−	+	−	+
Kerschbaum & Schröpfer, '14	−	−	+	++	++
Chenette et. al, '15	+	+	+	−	++

Wir verzichten an dieser Stelle aus Platzgründen auf detaillierte Beschreibungen der Verfahren, die wir nicht umsetzen. Um trotzdem eine Begründung für deren Ausschluss zu geben, folgt eine kurze Erläuterung der jeweils entscheidenden praktischen Nachteile, auf die die Originalautoren in ihren zumeist theoretischen Abhandlungen nicht oder nur oberflächlich eingehen.

Die Autoren von [KaAK10] und [LCYJ+14] setzen die detaillierte Kenntnis der zu verschlüsselnden Klartextdaten vor der Verschlüsselung voraus, (insbesondere Minimal- und Ma-

[2] die Autoren nehmen keine Sicherheitsanalyse ihres Verfahrens vor

ximalwerte), was problematisch für Datensätze ist, die mit der Zeit unvorhersehbar wachsen. Außerdem ist deren zu speichernder Zustand im Vergleich zu anderen OPE-Verfahren relativ groß. Der Ansatz von [LiWa12] benötigt im Voraus sogar die minimale Differenz zwischen zwei zu verschlüsselnden Klartextwerten, um bei der Verschlüsselung ein zufälliges, aber nicht zu großes Rauschen einzustreuen. Dieses Vorgehen ist aber insbesondere für den Einsatz mit Gleitkommazahlen problematisch, da deren Abstand zueinander theoretisch beliebig klein werden kann. Das Verfahren von [PoLZ13] benötigt, wie bereits erwähnt, eine zusätzliche Komponente ("OPE Server"), die auf Datenbankseite läuft und sich um Neuverschlüsselungen kümmert (vgl. Abschnitt 3.1-I). Die Installation einer solchen Komponente ist bei Clouddatenbank-Anbietern in der Regel nicht vorgesehen. Der Ansatz von [CLWW15] sieht keine Entschlüsselung vor, sondern bietet nur einen Vergleichsoperator für die Schlüsseltexte.

Wir konzentrieren uns also auf die Verfahren von [WRGA+13] und [KeSc14], für die wir im Folgenden jeweils die Funktionsweisen, sowie ihre praktischen Schwächen und unsere Modifikationen zu deren bestmöglichen Ausgleich beschreiben. Als praktische Referenz untersuchen wir außerdem das Verfahren von [BoCO11], wobei es sich um eine leichte Weiterentwicklung von [BCLO09] handelt, dem einzigen bisher praxisrelevanten OPE-Verfahren, welches in [PRZB11] und [TKMZ13] zum Einsatz kommt. Es bietet zudem als einziger zustandsloser Ansatz den Vorteil, dass kein Index zur Speicherung von Klartext-Schlüsseltext-Paaren vorgehalten werden muss. Das erleichtert den Einsatz in Client-Server Anwendungen, wie er in Datenbank-Szenarien üblich ist.

3.2.1 Random Subrange Selection ([WRGA+13])

Beschreibung

Die Autoren führen drei OPE-Verfahren ein, namentlich die "Random Offset Addition"(ROA), das "Random Uniform Sampling"(RUS) und die "Random Subrange Selection"(RSS). Da ROA trivial ist und von einem Angreifer bereits mit der Kenntnis nur eines einzigen Klartext-Schlüsseltext-Paares gebrochen werden kann, gehen wir darauf nicht weiter ein, sondern behandeln RSS mit RUS als dessen Teilprozedur.

RSS kann wie folgt beschrieben werden. Zunächst wird zufällig entschieden, wie die Unter- und Obergrenze r_{min} und r_{max} des Schlüsseltextbereichs R ermittelt werden, entweder durch $r_{min} \in [1, |R|-|D|+1]$ und $r_{max} \in [r_{min}+|D|-1, |R|]$ oder durch $r_{max} \in [|D|, |R|]$ und $r_{min} \in [1, r_{max} - |D| + 1]$. Innerhalb dieser Intervalle wird der entsprechende Wert jeweils zufällig gezogen. Danach wird mit Hilfe eines alternativen OPE-Verfahrens eine ordnungserhaltende Funktion f (im folgenden kurz: OEF) von $D = [1, |D|]$ nach $R = [1, r_{max} - r_{min} + 1]$ erzeugt. Dazu benutzen wir RUS, wie im folgenden Absatz beschrieben. Zum Schluss wird $r_{min} - 1$ zu allen Schlüsseltexten addiert.

RUS wird mit der OEF f initialisiert, in der nur Klartext-Schlüsseltext-Paare für die kleinsten und größten Werte von D und R (wie zuvor durch RSS bestimmt) enthalten sind. Eine rekursive Sample-Prozedur wählt dann zufällig ein Element $p \in [d_{min}, d_{max}]$ und $c \in [r_{min} + p - d_{min}, r_{max} + p - d_{max}]$. Damit spaltet p den Klartextbereich D und c den Schlüsseltextbereich R in je zwei Teilbereiche. Das Paar (p, c) wird zu f hinzugefügt und die Sample-Prozedur fährt für die jeweils neuen Teilbereiche von D und R rekursiv fort wie zuvor, bis alle Werte aus D einen Schlüsseltext zugewiesen bekommen haben.

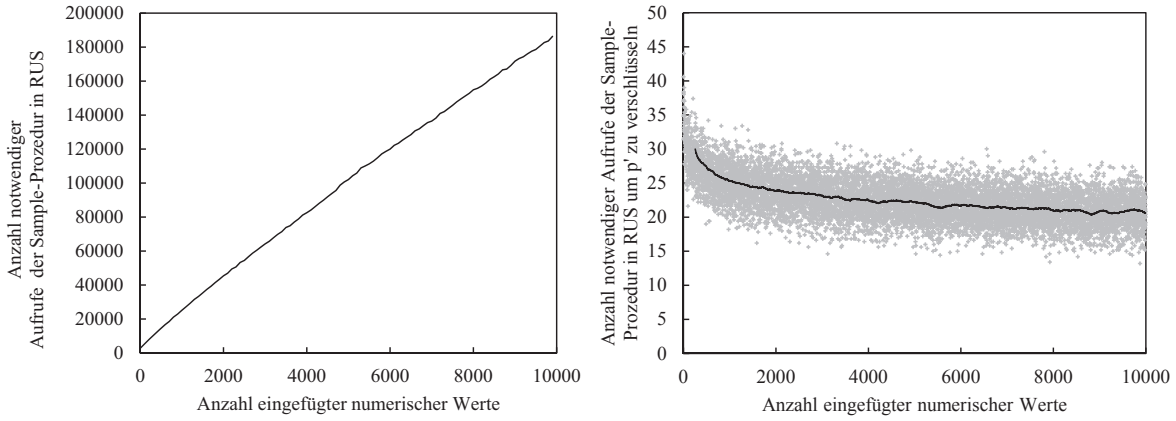

Abb. 1: Benötigte Samplings von RUS insgesamt (links) und pro Verschlüsselung (rechts)

Praktische Schwächen

RSS und RUS haben zwei Schwächen. Zum einen können nur positive Zahlen verschlüsselt werden. Zum anderen muss immer der gesamte Klartextbereich D verschlüsselt werden, statt gezielt nur die gewünschten Werte zu verschlüsseln (vgl. Abschnitt 3.1-IV).

Unsere Modifikationen

Die erste Schwäche lässt sich trivial beseitigen, indem die Sample-Prozedur von RUS mit einem entsprechenden negativen Wert für d_{min} anstelle von 1 initialisiert wird. Das erweitert den Klartextbereich D in die negativen Zahlen. Da das Verfahren nur auf zufälligen Ziehungen innerhalb bestimmter Intervalle sowie einigen Additionen und Subtraktionen beruht, beeinflusst das nicht seine Funktionsweise. Um der Notwendigkeit jeweils den gesamten Klartextbereich D verschlüsseln zu müssen, entgegenzuwirken, modifizieren wir RSS und RUS wie folgt.

Zunächst definieren wir p' als den Klartextwert, den wir verschlüsseln wollen. Wir modifizieren die Sample-Prozedur von RUS, indem wir ihr einen Parameter für p' hinzufügen. Statt die Sample-Prozedur nach einem Split des Klartextbereichs nun immer rekursiv für die beiden neuen Teilbereiche $[d_{min}, p-1]$ und $[p+1, d_{max}]$ durchzuführen, tun wir dies nur für den niederen Teilbereich, falls $p' \in [d_{min}, p-1]$, oder nur für den höheren Teilbereich, falls $p' \in [p+1, d_{max}]$. Dies reduziert die Anzahl der notwendigen Aufrufe der Sample-Prozedur von $|R|$ auf $log_2(|R|)$.

Dann modifizieren wir RSS selbst. Statt mit dem kompletten Klartextbereich D zu beginnen, wird die Sample-Prozedur von RUS nur mit dem Teilbereich $[d_1, d_2]$ initialisiert, wobei d_1 der größte bereits verschlüsselte Wert kleiner p' und d_2 der kleinste bereits verschlüsselte Wert größer als p' ist. Während mehr und mehr Werte verschlüsselt werden, senkt dies die durchschnittlich notwendige Anzahl der Ausführungen der Sample-Prozedur zusätzlich (siehe Abbildung 1, rechts). Damit dies aber auch schon für das erste p' funktioniert, nachdem r_{min} und r_{max} während der Initialisierung von RSS ermittelt wurden, müssen zu Beginn die kleinsten und größten Paare (p_{min}, c_{min}) und (p_{max}, c_{max}) zu f hinzugefügt werden. Die Sample-Prozedur bestimmt dafür c_{min} aus $[r_{min}, r_{max} - 1]$ und c_{max} aus $[c_{min} + 1, r_{max}]$.

Abbildung 1 illustriert die durchschnittlichen Verbesserungen an einem Beispiel, für das 20 mal 10000 zufällig gezogene 32-Bit Integer-Werte verschlüsselt wurden. Es zeigt dabei die Anzahl notwendiger Ausführungen der Sample-Prozedur insgesamt und pro Verschlüsselung. Um die-

se 10000 Werte zu verschlüsseln, werden anstelle der ursprünglich $|D| = 2^{32} = 4294967296$ Samplings im Schnitt nur 186.287 Samplings (also nur 0.004%) benötigt. Unsere Implementation braucht dafür weniger als eine Sekunde. Diese Anzahl ist natürlich kleiner, wenn weniger verschlüsselte Werte benötigt werden (z.B. nur 2765 im Durchschnitt für 100 Werte). Außerdem sinkt wie bereits angedeutet die notwendige Anzahl von Samplings pro Verschlüsselung, je mehr Werte bereits verschlüsselt worden sind (im Beispiel von den erwarteten $log_2(|R|) = log_2(2^{32}) = 32$ bei Beginn auf 21 bei der 10000. Verschlüsselung).

3.2.2 Optimal Average-Complexity Ideal-Security OPE ([KeSc14])

Beschreibung

Das OPE-Verfahren von [KeSc14] kann wie folgt beschrieben werden. Die OEF f wird initialisiert mit zwei Klartext-Schlüsseltext-Paaren: $(-1,-1)$ und $(|D|,|R|)$. Neue Verschlüsselungen (p,c) werden immer zwischen (p_n,c_n) und (p_{n+1},c_{n+1}) eingefügt, wobei $p_n \leq p < p_{n+1}$ und $c = c_n + \lceil \frac{c_{n+1}-c_n}{2} \rceil$. Falls $p = p_n$ ist, wurde p bereits verschlüsselt. Falls $c_{n+1} - c_n = 1$ ist, dann ist an der notwendigen Position in R kein Platz mehr, um ein neuen Schlüsseltext c aufzunehmen. In diesem Fall kommt es zu einer Neuverschlüsselung aller bereits verschlüsselten Klartexte, die wie folgt funktioniert: mit allen bereits verschlüsselten und sortierten Klartexten $p_1...p_m$ wird von neuem begonnen mit $p = p_{\lfloor \frac{m}{2} \rfloor + 1}$ und dann rekursiv in den Intervallen $p_1...p_{\lfloor \frac{m}{2} \rfloor}$, falls $m > 1$ bzw. $p_{\lfloor \frac{m}{2} \rfloor + 2}...p_m$, falls $m > 2$.

Praktische Schwächen

Die größte Schwäche des OPE-Verfahrens von [KeSc14] ist die Phase der Neuverschlüsselung, denn diese bedeutet in der Praxis das Auslesen und Zurückschreiben aller bereits verschlüsselten Werte. Um die Anzahl des Auftretens solcher Phasen so gering wie möglich zu halten, sollte R so möglichst groß gewählt werden. Eine Grenze stellen hier die zur Verfügung stehenden Datentypen dar, die das jeweilige Datenbanksystem anbietet. Mit einer Klartextlänge von n Bit empfehlen die Autoren eine Schlüsseltextlänge von λn Bit, wobei $\lambda = 6.31107$, um die Wahrscheinlichkeit für das Auftreten einer Neuverschlüsselung auf $1/n$ zu senken. Sie zeigen in praktischen Experimenten aber auch, dass bereits mit $\lambda = 3$ (teilweise sogar $\lambda = 2$) für viele Datensätze kaum Neuverschlüsselungen nötig sind. Eine andere praktische Schwäche ist die Tatsache, dass die Anzahl der auftretenden Neuverschlüsselungsphasen auch von der Reihenfolge abhängig ist, in der die Werte des Datensatzes verschlüsselt werden. Den besten Fall stellt das Einfügen aller Elemente eines perfekt balancierten binären Suchbaums in Pre-Order-Traversierung dar. Der durchschnittliche Fall ist eine rein zufällige Reihenfolge. Im schlimmsten Fall sind die zu verschlüsselnden Werte bereits vorsortiert. Darüber hinaus ist das Verfahren für negative Werte nicht definiert.

Obwohl der Algorithmus von [KeSc14] also einige Schwächen besitzt, ziehen wir ihn trotzdem für den praktischen Einsatz in Betracht, denn er basiert auf sehr simplen Berechnungen, die nicht einmal Zufallselemente beinhalten. Werden vorsortierte Eingaben vermieden, sollte sich die Verschlüsselung im Datenbankbetrieb also kaum in der Laufzeit bemerkbar machen.

Unsere Modifikationen

Da wir die kostspieligen Phasen der Neuverschlüsselung nicht vermeiden können (außer durch eine entsprechend große Definition von R), beschränken wir uns auf die Modifikation der Initialisierung von f mit $(-|D|,-|R|)$ und $(|D|,|D|^{\lambda})$, anstelle von $(-1,-1)$ und $(|D|,|R|)$.

Dies erweitert D ähnlich unserer Modifikation von [WRGA+13] um negative Werte. R wird entsprechend vergrößert, damit die Anzahl auftretender Neuverschlüsselungen nicht steigt.

3.2.3 mOPE ([BoCO11])

Beschreibung

mOPE (modular OPE) ist eine Erweiterung des Verfahrens von [BCLO09], welches auf der Beobachtung basiert, dass jede ordnungserhaltende Funktion von $\{1...M\}$ nach $\{1...N\}$ als Kombination von M aus N geordneten Elementen repräsentiert werden kann. Schlüsseltexte können damit final so berechnet werden wie das Sampling aus einer hypergeometrischen Verteilung. [BoCO11] erweitert diesen Ansatz, indem vor der Verschlüsselung ein geheimer Offset zum Klartext addiert bzw. bei Entschlüsselung subtrahiert wird. Es gilt also $Enc_{mOPE}(x) = Enc(x + m)$ mit dem geheimen Offset m und $Dec_{mOPE}(x) = Dec_{OPE}(x) - m \bmod |D|$, wobei $|D|$ die Größe des Klartextbereichs ist. Dies erschwert das Abschätzen der Verteilung der Klartexte innerhalb von D.

Praktische Schwächen

Wie beschrieben ist der Kern dieses OPE-Verfahrens das Sampling aus einer hypergeometrischen Verteilung. Dies ist zum einen naturgemäß nur mit positiven Ganzzahlen möglich und zum anderen relativ rechenintensiv. Daher kann hier nur versucht werden, möglichst praktikable Laufzeiten zu erreichen. Die unseres Wissens schnellste Implementation von [BCLO09] findet sich in "CryptDB" [PRZB12]. Dort wird für das Sampling der H2PEC Algorithmus benutzt [KaSc88]. Mit unserer Java-Portierung von H2PEC erreichen wir je nach verwendeter Datenbank Verschlüsselungszeiten von unter 2 ms für einen 32-Bit Integer-Wert (im Vergleich zu 5 ms bei "CryptDB").

4 Implementation und Experimente

Da Zugriff und Speicherverwaltung von SFD spaltenbasiert erfolgen, legen wir die Indizes für [WRGA+13, KeSc14] in gleicher Weise an. Für unsere Experimente verschlüsseln wir bis zu 20000 innerhalb des Klartextbereichs gleichverteilte und zufällig ausgewählte Zahlen mit Apache Cassandra (Version 3.0.2) und HBase (Version 1.1.2) als darunterliegende Datenbanken. Dazu nutzen wir die drei OPE-Verfahren wie in Abschnitt 3.2 beschrieben. Um praxisnah zu sein, verschlüsseln wir 32-Bit Integer-Werte. Während für [BoCO11] und [WRGA+13] die Reihenfolge der verschlüsselten Werte keine Rolle spielen, so betrachten wir für [KeSc14] auch die drei verschiedenen Fälle wie in Abschnitt 3.2.2 beschrieben. Wir benutzen lokale Installationen der Datenbanken, da wir an den Laufzeiten der Algorithmen in Kombination mit Schreib-/Lesegeschwindigkeiten interessiert sind, die nicht durch Netzwerkeffekte verzerrt werden sollen. Alle Implementationen[3] erfolgten in Java 8. Alle Messungen wurden durchgeführt auf einer Intel Core i7-4600U CPU @ 2.10GHz, 8GB RAM, einer Samsung PM851 256GB SSD auf Ubuntu 15.04.

Abbildung 2 zeigt die Resultate. Dargestellt ist jeweils der Mittelwert aus mindestens zehn Messungen. Obwohl sie einen Index verwalten müssen, sind die Verfahren von [WRGA+13, KeSc14] grundsätzlich schneller als das zustandslose Schema von [BoCO11]. Eine Ausnahme bildet nur die Anwendung des Verfahrens von [KeSc14] mit vorsortiertem Input, die gänzlich

[3] verfügbar unter `https://github.com/dbsec/FamilyGuard`

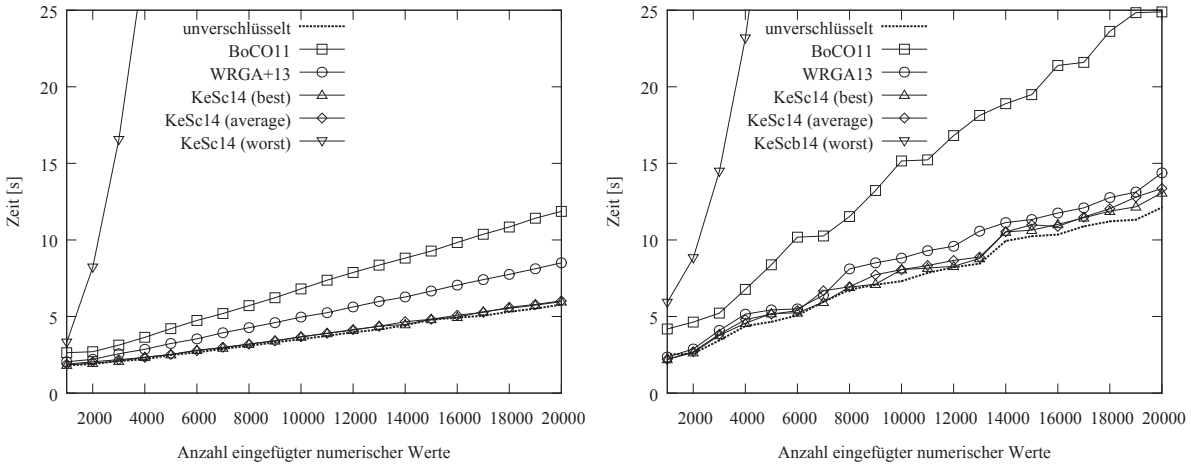

Abb. 2: Verschlüsselungsdauer unter Cassandra (links) und HBase (rechts)

vermieden werden sollte. Der durch die Verschlüsselung bedingte Geschwindigkeitsverlust beträgt im besten Fall gerade einmal 3% im Durchschnitt mit [KeSc14] (best und average) und Cassandra. Mit Ausnahme von [KeSc14] (worst) dauert das Einfügen mit Verschlüsselung im schlimmsten Fall etwa doppelt solange wie ohne. Zudem ist Cassandra verglichen zu HBase im Schnitt etwa um 40% schneller. Die Erklärung hierfür liegt darin, dass [WRGA$^+$13] und [KeSc14] so schnell sind, dass die Zeit der reinen Einfügeoperation in die Datenbank einen signifikanten Teil der Gesamtzeit der Verschlüsselung ausmacht. Da Cassandra auf Schreibvorgänge optimiert ist, zeigt sich hier der entsprechende Vorteil. Eine Ausnahme bildet der schlechteste Fall von [KeSc14], bei dem neben den Schreibvorgängen auch oft aus der Datenbank gelesen werden muss. Dies schlägt sich in einem ca. 12-15%igen Vorsprung für HBase nieder, das im Gegensatz zu Cassandra auf Lesevorgänge optimiert ist.

Da die Entschlüsselung sehr simpel ist, verzichten wir auf entsprechende Abbildungen. Im Fall von [WRGA$^+$13] und [KeSc14] besteht sie aus einem Nachschlagevorgang im entsprechenden Index. Dies dauert auch bei großen Datensätzen in der Regel weniger als 1 ms. Das nicht zustandsbasierte Verfahren von [BoCO11] kann auf einen solchen Index nicht zurückgreifen. Die notwendige Berechnung benötigt unabhängig von der Datensatzgröße im Schnitt 5 ms.

5 Diskussion

Die Ergebnisse zeigen, dass OPE in SFD Datenbanken effizient betrieben werden kann. Es ist jedoch lohnenswert, sich je nach Anforderungen des späteren Datenbankbetriebs für ein bestimmtes Verfahren zu entscheiden. Möchte man in jedem Fall den zusätzlichen Speicherplatzbedarf für die Sicherung eines Wörterbuchs vermeiden, so kommt nur [BoCO11] in Frage. Ist nicht zu befürchten, dass der Input bereits stark vorsortiert ist und man Wert auf Geschwindigkeit legt, sollte man auf [KeSc14] zurückgreifen. [WRGA$^+$13] ist ein Kompromiss aus beiden Verfahren. Es benötigt zwar ein Wörterbuch, ist aber trotzdem ausreichend schnell und unabhängig vom Input. Die Gesamtperformance des Datenbanksystems lässt sich unter Umständen zusätzlich steigern, indem je nach Anforderung für verschiedene Spalten verschiedene OPE-Verfahren eingesetzt werden. Für die Verschlüsselung der Zeilen-Identifikatoren können jedoch nur [BoCO11] oder [WRGA$^+$13] in Betracht gezogen werden, da deren Schlüsseltexte unveränderlich sind (vgl. Abschnitt 3.1-I).

6 Verwandte Arbeiten

Wie eingangs erwähnt, gibt es derzeit kaum Arbeiten über den praktischen Einsatz von OPE im Zusammenspiel mit Datenbanken. Eine der wenigen Implementierungen findet sich im bereits erwähnten "CryptDB", in dem das Verfahren von [BCLO09] zum Einsatz kommt. Auch "Monomi" [TKMZ13] benutzt diesen Ansatz. Beide Systeme sind jedoch für die Verschlüsselung SQL-basierter Datenbanken konzipiert.

7 Zusammenfassung und Ausblick

Wir haben gezeigt, wie OPE in NoSQL SFD verwendet werden kann und quantifizierten und optimierten die Laufzeit von drei konkreten Verfahren in Kombination mit den zwei derzeit populärsten SFD. Derartige Untersuchungen haben wir auch bereits für Verfahren zu durchsuchbaren Verschlüsselung durchgeführt. Unser nächstes Ziel ist daher einen Proxy-Client ähnlich zu "CryptDB" für SFD zu entwickeln, der es ermöglicht, auch komplexere Anfragen über verschlüsselte Daten auszuführen.

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

Literatur

[AKSX04] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu: Order preserving encryption for numeric data. *In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ACM (2004), 563–574.

[BCLO09] A. Boldyreva, N. Chenette, Y. Lee, A. O'Neill: Order-preserving symmetric encryption. *In: Advances in Cryptology-EUROCRYPT 2009*, Springer (2009), 224–241.

[BGSM+11] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon: Apache Hadoop goes realtime at Facebook. *In: Proceedings of the SIGMOD International Conference on Management of Data*, ACM (2011), 1071–1080.

[BoCO11] A. Boldyreva, N. Chenette, A. O'Neill: Order-preserving encryption revisited: Improved security analysis and alternative solutions. *In: Advances in Cryptology–CRYPTO 2011*, Springer (2011), 578–595.

[CDGH+08] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes: Bigtable: A distributed storage system for structured data. In: *ACM Transactions on Computer Systems (TOCS)*, 26, 2 (2008), 4.

[CLWW15] N. Chenette, K. Lewi, S. A. Weis, D. J. Wu: Practical Order-Revealing Encryption with Limited Leakage (2015).

[Harr15] G. Harrison: Database Survey. *In: Next Generation Databases*, Springer (2015), 217–228.

[KaAK10] H. Kadhem, T. Amagasa, H. Kitagawa: MV-OPES: Multivalued-order preserving encryption scheme: A novel scheme for encrypting integer value to many different values. In: *IEICE TRANSACTIONS on Information and Systems*, 93, 9 (2010), 2520–2533.

[KaSc88] V. Kachitvichyanukul, B. W. Schmeiser: Algorithm 668: H2PEC: sampling from the hypergeometric distribution. In: *ACM Transactions on Mathematical Software (TOMS)*, 14, 4 (1988), 397–398.

[KeSc14] F. Kerschbaum, A. Schröpfer: Optimal average-complexity ideal-security order-preserving encryption. *In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM (2014), 275–286.

[LaMa10] A. Lakshman, P. Malik: Cassandra: a decentralized structured storage system. In: *ACM SIGOPS Operating Systems Review*, 44, 2 (2010), 35–40.

[LCYJ+14] Z. Liu, X. Chen, J. Yang, C. Jia, I. You: New order preserving encryption model for outsourced databases in cloud environments. In: *Journal of Network and Computer Applications* (2014).

[LiWa12] D. Liu, S. Wang: Programmable order-preserving secure index for encrypted database query in service cloud environments. *In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, IEEE (2012), 502–509.

[MCOK+15] C. Mavroforakis, N. Chenette, A. O'Neill, G. Kollios, R. Canetti: Modular Order-Preserving Encryption, Revisited. *In: Proceedings of the 2015 ACM SIGMOD Int. Conference on Management of Data*, ACM (2015), 763–777.

[OGOGG+11] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, J. Abramov: Security issues in nosql databases. *In: Trust, Security and Privacy in Computing and Communications, 2011 IEEE 10th International Conference on*, IEEE (2011), 541–547.

[PoLZ13] R. A. Popa, F. H. Li, N. Zeldovich: An ideal-security protocol for order-preserving encoding. *In: IEEE Symp. on Security and Privacy* (2013), 463–477.

[PRZB11] R. A. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan: CryptDB: protecting confidentiality with encrypted query processing. *In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, ACM (2011), 85–100.

[PRZB12] R. A. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan: CryptDB: Processing queries on an encrypted database. In: *Communications of the ACM*, 55, 9 (2012), 103–111.

[RACY15] D. Roche, D. Apon, S. G. Choi, A. Yerukhimov: POPE: Partial order-preserving encoding. Tech. Rep., Cryptology ePrint Arch. 2015/1106 (2015).

[TKMZ13] S. Tu, M. F. Kaashoek, S. Madden, N. Zeldovich: Processing analytical queries over encrypted data. *In: Proceedings of the VLDB Endowment*, VLDB Endowment (2013), Bd. 6, 289–300.

[WRGA+13] S. Wozniak, M. Rossberg, S. Grau, A. Alshawish, G. Schaefer: Beyond the ideal object: towards disclosure-resilient order-preserving encryption schemes. *In: Proceedings of the 2013 ACM workshop on Cloud computing security*, ACM (2013), 89–100.