

FLAM®

Multifunktionstool (seit 1.1.1986 im Einsatz)

Standard
im deutschen
Bank- und Börsenwesen
Kartenabrechner/-hersteller
SCHUFA

Hersteller

limes datentechnik® gmbh

alles aus einer Hand:

Forschung / Entwicklung
Betreuung / Beratung
hervorragender Service (Hotline)
zentral gesteuerter Vertrieb

direkte Kommunikation
zwischen Hersteller und Anwender/-gruppen
Partnerschaft mit allen Herstellern (User Groups etc.)

FLAM®

Was heißt das?

F rankenstein*

L imes®

A ccess

M ethod

Wir *FLAM*®-bieren Daten.

keine Derivate von Dritten

keine divergenten Entwicklungen / keine Kompatibilitätsprobleme

keine Probleme bzgl. Zuordnung eines Ansprechpartners

limes®: Leistung im Grenzbereich des Machbaren.

*) Großvater des Firmengründers

FLAM®

Headlines

verdichten
verschleiern
verschlüsseln
legendieren
versiegeln
konvertieren
konkartinieren
splitten
zugreifen
selektieren
automatisieren
archivieren

...

homogen wie heterogen kompatibel
auf allen System-Plattformen
Versionen sind abwärts-/seitwärtskompatibel

Key Handling / Management

Ein außergewöhnliches Produkt, das sehr viele Probleme löst!

Datenvolumen

Ein guter Kompressor erkennt die Redundanz der Syntax und sorgt dafür, dass diese Daten gegen NULL reduziert werden (=> konvergierend, d.h. wie in einer limes-Funktion den Grenzwert bilden). Der Kompressor muss diese Syntax nicht wirklich kennen, sondern dynamisch, adaptiv erkennen. Es verbleiben weitgehend nur die Nettodaten, die nach mathematischen Verfahren verlustfrei (engl.: lossless) komprimiert werden.

Diese Daten werden kontextuell, strukturell, nach Häufigkeiten etc. so komprimiert, dass das komprimierte Datenvolumen bei gleichen Inhalten in etwa gleich groß bleibt (=> invariantes Datenvolumen => Komprimat). Man kann z.B. SWIFT- oder XML-Dateien bestens komprimieren, ohne zu wissen, dass es SWIFT / XML ist.

Sehr große Dateien können trotz Komprimierung z.B. für den File Transfer (Dauer der störungsfreien Übertragung) oder bestimmte Datenträger zu groß bleiben. Es kann demnach zweckmäßig sein, solche Dateien (=> Komprimat) seriell oder – durchaus auch aus Sicherheitsgründen – parallel in Teilen zu splitten, und zwar "wraparound" in Einheiten (Teil 1,2,...,n; 1,2, ...,n; 1,2, ...) von 4 Bytes (Speicherworte a 32 Bits).

Man kann Dateien archivieren, in dem man das Komprimat parallel in 3/3 aufteilt und jedes Teil kopiert. Man legt Teil 1+2 nach A, Teil 1+3 nach B und Teil 2+3 nach C. Damit dürften Daten an 1 Standort verlorengehen, an 1 Standort allein kann man die Daten nicht rekonstruieren (=> perfekter Schutz vor Zugriffen, die bezogen auf den einzelnen Standort nicht zugelassen sind). Eine von einem Krypto-Verfahren nebst Key Management abhängige kryptographische Sicherung der Daten eines Archivs über sehr viele Jahre hinweg wäre unnötig.

Durch das invariante Datenformat, das man mit dem Kompressor bilden kann, spielt es keine Rolle, auf welcher System-Plattform die archivierten Daten in der Zukunft gelesen, d.h. dekomprimiert werden.

Protokolle

Das Komprimat kann beliebig transportiert werden. Es entsteht ein einfacher Bit-Strom (binary file), den man nach BASE64 konvertieren könnte. Man kann eine beliebig formatierte Datei erzeugen, deren Format vom Original resp. späteren Dekomprimat unabhängig ist. Es muss möglich sein, Formate zu konvertieren:

Original ⇔ Komprimat ⇔ Dekomprimat.

Das Komprimat ist so aufgebaut, dass ein Bit-Fehler, der ggf. unerkant bleibt, oder das Fehlen resp. das unerlaubte Hinzufügen von Daten zu einem schweren Fehler bei der Dekomprimierung führen muss. Damit ist jede Manipulation ausgeschlossen. Die Daten sind allein dadurch in gewisser Weise versiegelt.

Es gibt keine Probleme mit Codierungen (ASCII, EBCDIC). Das betrifft auch das Problem der Alphabetisierung. Das darf nicht Gegenstand des Datenaustausches über Leitungen, Internet oder per Datenträger sein.

Der De-/Kompressor hilft durch Zeichen-Konvertierung, diese Probleme zu lösen, z.B. durch entsprechende Funktionen oder User-Exits zur individuellen Anbindung solcher Funktionen (z.B. UTF8). Jede Konvertierung ist letztlich über ein Exit-Modul realisierbar. Und für Zeichen-Konvertierungen 1:1 wie EBCDIC ⇔ ASCII mit unterschiedlichen Tabellen muss nur die gewünschte Tabelle zugewiesen werden. Dafür kann selbst innerhalb eines Alphabets wie EBCDIC ⇔ EBCDIC oder ASCII ⇔ ASCII Bedarf bestehen. Beim Lesen am Monitor und Drucken darf nicht übersehen werden, dass ggf. verschiedene Standards zugewiesen wurden, so dass der Eindruck entsteht, es wären Fehler im Dekomprimat. Das sind verschiedene Interpretationen. Mit dem Dekomprimat und dessen Codierung hat das nichts zu tun.

Heterogene Kompatibilität

Das bedeutet: => Invarianz der beteiligten System-Plattformen resp. Benutzung durch Applikationen.

Der Sender muss nicht wissen, welche System-Plattform der Empfänger benutzt. Nur der De-/Kompressor muss gleich sein. Das ist gerade im globalen Datenaustausch von großer Wichtigkeit, z.B. bei SEPA oder im globalen Clearing von "physical resp. billing records" der Telefongesellschaften und Provider etc.

Aber das allein sollte kein Grund sein, bei der Qualität eines Kompressors an Auswahlkriterien zu sparen. Die Kompressoren, die ihre Wurzeln im PC-Umfeld haben, verzichten auf Funktionen, die für Anwender in einem globalen Netz von großer Bedeutung sind. In sternförmigen Netzen muss der Betreiber des "Sterns" entscheiden, was gebraucht wird, denn die Hauptlast liegt bei ihm. Das gilt betriebswirtschaftlich auch für den Verbrauch an Ressourcen (CPU-Zeit, Speicher). Ein 2-Schritt-Verfahren mit einem temporären Zwischenergebnis beim Komprimieren behindert Ablauf sowie Applikationen und ist teuer. Der Kompressor muss in der Lage sein, straight forward zu arbeiten, sonst ist er für viele Anwendungen völlig ungeeignet. Man komprimiert/dekomprimiert möglichst früh resp. spät, d.h. mit oder dicht an den entscheidenden Applikationen.

Man kann effizient - offline - ver-/entschlüsselt arbeiten, und zwar auf den Systemen, auf den/en die Applikation/en ohnehin mit klaren Daten abläuft. Die beteiligten Server sind ggf. wieder ganz andere Systeme. Die können diese Daten nicht lesbar machen und sind dadurch gegenüber dem Zugang vom externen Netz besonders gesichert. Der Datenaustausch komprimierter/verschlüsselter Daten am Server erfolgt intern.

In Verbindung mit Kryptographie muss der Anwender abdruckbare Schlüssel (lateinisches Alphabet) benutzen dürfen, so dass selbst hier zwischen ASCII und EBCDIC Key-Kompatibilität geschaffen werden muss.

Revisionssicherheit

Der Kompressor muss Komprimat so aufbauen, dass er den Input so reproduzieren kann, wie er bereitgestellt wurde. Es darf z.B. keine Korrekturen geben, auch wenn die Input-Datei "Fehler" in deren Syntax enthält. Das gilt auch für Optimierung im Original/Dekomprimat. Das sind ggf. Features – aber nicht Standard.

Wird der Zeichensatz beim Komprimieren konvertiert, muss er beim Dekomprimieren analog durch Re-Konvertierung reproduzierbar sein. Abweichungen von solchen Regeln muss der Anwender individuell per Parameter o.a. vorgeben und die Folgen davon selbst vor der Revision (ggf. staatl. Aufsicht) verantworten.

Das Dekomprimat auf einer anderen System-Plattform muss sich so eng wie möglich am Input (Original) orientieren (Default). Ggf. ist sogar eine Format-Emulation vom Host auf PC oder umgekehrt sinnvoll.

Das Komprimat sollte möglichst ein Unikat sein. Wiederholt man die Komprimierung derselben Input-Datei, sollte das Komprimat anders (modifiziert) codiert werden. Damit kann der Vorgang der Erstellung des Komprimats auch beim Austausch zwischen verschiedenen Systemen eindeutig zugeordnet werden. Das hat keine Auswirkungen auf die Größe des Komprimats und schon gar nicht auf deren dekomprimierten Inhalt.

Ein Komprimat - als Unikat - läßt sich beim Bilden der Checksummen auf einfache Weise verschleiern. Die Streuung der Bytes im Komprimat ist dadurch gleichmäßig - unabhängig vom Ergebnis der Komprimierung.

Man kann komprimierte/verschlüsselte Daten legendieren, so dass einem "Beobachter" nicht sofort auffällt, was das für Daten sein könnten. Das erschwert Angreifern die Suche nach Daten, um die es ihnen geht.

Kryptographie

Allein durch das reduzierte Datenvolumen kann man die komprimierten Daten effizienter verschlüsseln. Man kann das Komprimat "in sich" mit einem einheitlichen Standard wie **AES** und einem beliebig großen Zufalls-Schlüssel perfekt verschlüsseln und mittels hierarchischer MAC-Techniken perfekt signieren.

Nur der Zufalls-Schlüssel selbst muss mit dem vereinbarten Verfahren nebst den geheimen Schlüsseldaten verschlüsselt und im Header des Komprimats signiert abgelegt werden. Damit kann man diesen Teil beliebig oft kryptographisch sicher und wie vereinbart modifizieren. Das mit **AES** verschlüsselte Komprimat bleibt völlig unverändert. Das vereinbarte Verfahren nebst zugeordnetem Key Management wird – falls es Unterschiede gibt - auf diese Weise kompatibel gemacht, auch wenn man eine Datei im internen Bestand nur umschlüsseln will. – Das gibt es u.U. in Archiven, wenn "alte" durch "neue" Standards ersetzt werden sollen.

So kann man auch zweckgebunden Dateien verändern, in dem man nur den Header kryptographisch sicher umcodiert: Archiv, Übertragung, Backup etc. Für jeden Zweck die gleiche Datei, aber andere Zugriffsschlüssel resp. Verfahren zur Verschlüsselung nebst Key Management für den Header incl. dessen Signierung.

Kryptographie kann so auf einfache Weise hybrid kompatibel realisiert werden. Es spielt dabei keine Rolle, wenn ein Partner zum Teil Hardware einsetzt und der andere nicht. Das muss dann emuliert werden. Solche Lösungen braucht man selbst, wenn zur **AES**-Verschlüsselung (vorerst) nur einseitig CPACF benutzt wird.

Man kann solche Daten effizient zum Download bereitstellen, so dass der Empfänger selbst den Zeitpunkt bestimmt, wann er die Daten abrufen. Man kann sie auch per Post per Datenträger einfach versenden, wenn eine Datenübertragung nicht notwendig ist oder keine Vorteile bringt sowie in einem Backup-Konzept.

Zugriffsmethoden

Ein Komprimat kann in autarken Segmenten erzeugt und index-sequentiell verwaltet werden. Das gilt sogar, wenn man diese Daten verschlüsselt. Es wird ohnehin je autarkem Segment ein MAC gebildet. Man muss im quasi Online-Zugriff nur das entschlüsseln/dekomprimieren, was wirklich gebraucht wird. Dazu kann man intelligente Suchverfahren benutzen. Das erhöht enorm die Effizienz (Kosten senken).

Die Bundesbank hat eine solche intelligente, sehr effiziente Verwaltung von SEPA-Beständen. Die Telekom verwaltet seit über 10 Jahren gigantische Datenbestände index-sequentiell über ein Subsystem, das wie ein I/O-Treiber effizient mit komprimierten Daten umgehen kann. Die Integration einer gemäß **AES** standardisierten Verschlüsselung wäre völlig unproblematisch.

Bei verschlüsselten Daten kann man das selektieren, was gebraucht wird, und verschlüsselt an den Anwender weiterleiten, der berechtigt ist, diese Daten zu lesen, d.h. offline zu entschlüsseln. Damit verhindert man, dass auf die klaren Daten durch Unberechtigte (z.B. intern) zugegriffen werden kann.

Auch bei sequentieller Verarbeitung kann man Segmente, die nicht gebraucht werden, übergehen resp. abbrechen, wenn keine weiteren Segmente mehr gebraucht werden. Ohne einen solchen Aufbau der Daten in Segmenten, die autark komprimiert/verschlüsselt werden, muss immer alles entschlüsselt/dekomprimiert werden. Bei kryptographisch gesicherten Daten wäre das ein völlig unnötiges Sicherheits-Risiko.

Man kann alle Abläufe voll automatisieren. Durch die Bildung autarker Segmente ist Backup-Restart möglich. Dateikompressoren mit übergreifenden Techniken (z.B. thesaurierende Wörterbücher) über alle Daten können das nicht.

Alternative Algorithmen

Ein guter Kompressor kann die Komprimierung unterdrücken. Wer sagt denn, dass man immer gleich Daten komprimieren will? Vielleicht sind die Daten bereits komprimiert oder lassen sich nicht besonders komprimieren. Das wäre verschenkte Zeit. – Es gibt Daten, die werden verlustarm (engl.: low-loss) komprimiert. Das sieht gewaltig aus, ist aber irreversibel, falls man alles zurückhaben will, was das Original war, z.B. Dokumente. Oder man will nur wertvolle Funktionen des Kompressors – wie die [AES](#) Encryption – nutzen, ohne zusätzlich die Komprimierung einzuschalten, z.B. für solche gescannten Dokumente (Pixel-Daten).

Außerdem muss der Kompressor abschätzen können, was sich lohnt, überhaupt komprimiert zu werden. Ein Kompressor, der in autarken Segmenten arbeitet, kann diese Entscheidung für jedes Segment treffen – unabhängig von den bereits verarbeiteten Daten (Segmenten). Da kann sich manches ändern, z.B. in Dump's.

Was sich lohnt, [WIE](#) komprimiert zu werden, weiß nicht immer der Kompressor, sonst müßte er zur Optimierung sehr viel an CPU-Zeit aufwenden, so wie manche Tools das auf PC machen. Es kann also im Kompressor selbst Alternativen geben (MODE-Funktionen), die man nur – passend zur Anwendung – benutzen muss.

Und wenn es einen Parameter zur Steuerung der Komprimierung mit Alternativen gibt, sollte man diesen in die Tests – je Anwendungsfall – einbeziehen. Die Entscheidung, was am besten ist, soll ja lange Zeit von Bestand sein. Sonst kann man sogar zu völlig falschen Grundsatzentscheidungen bei Vergleichen kommen. Es ist keinesfalls ratsam – u.U. gefährlich, Komprimierung mit Vorgaben aus Daten statisch zu verknüpfen. Die regelmäßige Anpassung solcher Vorgaben (Tab's, DIC's) kann Abhängigkeiten schaffen, die teuer werden.

Der Dekompressor einer dynamisch, adaptiven Technik arbeitet synchron zum Kompressor und findet von allein die Informationen, die er braucht, um das korrekte Dekomprimat zu erzeugen (abwärtskompatibel).

Typische Sonderfälle

Kompressoren, die beim Dekomprimieren Fehler melden, machen den Anwender in der Regel auf Fehler in der Benutzung ggf. sogar in seiner eigenen Organisation (oder der seiner Partner) aufmerksam.

Kompressoren, die insbesondere heterogen kompatibel sind und sehr vielen Anforderungen genügen müssen, haben andere Testanforderungen/-möglichkeiten. Die komprimierten Daten sind mehrfach abgesichert.

Es wird oft übersehen, dass Daten zur Fehlerbehandlung wie Dumps, Protokolle etc. sensible Daten umfassen können. Da sollte man besser komprimieren und verschlüsseln, ehe man solche Daten weitergibt.

Umgekehrt kann man Auslieferungen komprimiert und verschlüsselt zum Download bereitstellen, so dass nur der berechtigte Empfänger damit etwas anfangen kann. Man nimmt z.B. einen mit Herstellern vereinbarten Vertrags-/Lizenz-/Installations-Schlüssel zur **AES** Encryption. Diesen Schlüssel kennt der Empfänger.

Wer Druckdaten erst aufbereitet und dann komprimiert, freut sich umsonst über den Komprimierungseffekt. Er hätte besser daran getan, erst zu komprimieren, dann die Daten zu versenden, am Empfangssystem wieder zu dekomprimieren und erst dort die Drucklisten druckbereit zu machen. – Das gilt insbesondere, wenn solche Daten obendrein verschlüsselt übertragen werden sollen, wie man das bei Dumps tun muss.

Ein "billiger" Kompressor ist oft überhaupt nicht billig. Er frisst CPU-Zeit, die an anderer Stelle zu Kosten führt, die der (Test-)Anwender, der komprimiert, gar nicht sieht. Von mangelnden Funktionen ganz abgesehen. Auf PC gibt es eine Fülle von Kompressoren, die im Kleinen etwas besser sind. Sie müssen arithmetisch komprimieren (so heisst das) und das kostet bei Massendaten enorm viel Geld – weil es CPU-Zeit frisst. Im Zweifel trifft es den, der in alle Richtungen mit solchen Daten kommunizieren muss (Zentrum).

FLAM®

Lizenzpolitik

Konzernlizenz

Lizenz für System-Plattform

Lizenz für Applikation

Lizenz für Partner / Kunden

Verbandslizenzen o.dgl.

Einzellizenzen

mit / ohne zeitliche Begrenzung

Wartung

Beratung / Betreuung

auch im techn. Umfeld

7 x 24 Stunden (Hotline)

Updates wg. Änderungen im Systemumfeld

Folgeversionen mit neuen Funktionen

Weiterentwicklung in Absprache mit Kunden

unverbindlich **kostenlose** Testversion (30 Tage)