Projektspezifikation
Software-/Backuplösung für ***arcun***

1 Zusammenfassung

Die Software-/Backuplösung für ***arcun*** muss in der Lage sein, das auf einem File-Server im Archiv in Dateiform liegende Archivgut auf die FAST-Speicherlösung zu kopieren, die notwendigen Metainformationen festzuhalten und den Zugang zu und die Kontrolle der ausgelagerten Dateien sicherzustellen. Dazu sollen die Dateien zusammen mit Metadaten im XMP-Format in einen ZIP-Container verpackt werden. Die vom FAST-System generierte Access-ID und File-GUID müssen lokal im Filesystem sowie in einer eigenen Datenbank gespeichert werden. Die GUID (ein Hashwert) dient dabei gleichzeitig zur Verifikation der Integrität der Dateien. Aus Sicherheitsgründen sollte eine Verschlüsselung der ausgelagerten Dateien erwogen werden.

2 Problemstellung

Wir können im Augenblick und für die an ***arcun*** beteiligten Archive davon ausgehen, dass im Archiv das digitale Archivgut auf einem File-Server liegt. Weil noch kein eigentliches Archivsystem für digitale Daten zur Verfügung steht, widerspiegelt die Ablagestruktur auf dem File-Server die Organisation des digitalen Archivs. In gewissen Fällen wird die Ablagestrukturinformation auch in den Katalog (das AIS) als Pfad und Dateiname eingetragen und erlaubt so den Zugriff auf die Datei aus dem digitalen Katalog.

In der Regel ist dieser File-Server auch gleich noch der *workspace* für den mit der digitalen Archivierung betrauten Mitarbeiter.

Mit ***arcun*** soll nun von dieser Ablagestruktur auf dem lokalen File-Server eine WORM[1] Kopie an einem ausgelagerten Ort angelegt werden. Damit wird die lokale Kopie zu einem echten *workspace*, fehlerhafte Eingriffe auf die lokalen Daten bleiben ohne Folge.

Dazu sollte aus Sicht der Archive der FAST Speicher so einfach wie ein File-Server zu bedienen sein. Durch den verschlüsselten Zugriff über das Internet ist die Zugriffsmethode aber eine völlig andere:

[1] WORM ist die Abkürzung für *write once read multiple (times)* oder *write once read many (times)*, siehe Wikipedia: http://de.wikipedia.org/wiki/WORM . Eine wesentliche Eigenschaft von WORM-Speichermedien ist, dass die Daten nicht mehr gelöscht werden können.
Bei der technischen Lösung von FAST wird die WORM-Eigenschaft des Datenspeichers mittels Soft- und Hardware implementiert.

Eine Datei wird über das FAST-API oder ein *stand-alone* Tool[2] vom Archiv zum FAST-Speichersystem übertragen. Der FAST Server quittiert die Einlieferung mit einer Access-ID und File-GUID. Die Access-ID ist der Zugriffschlüssel, um erneut auf die Datei zuzugreifen, entspricht also //servername/path/filename im Fileserverumfeld. Die Access-ID wird vom FAST-Speichersystem vergeben (dies im Gegensatz zum File-Server) und ist eindeutig innerhalb eines Mandanten. Die File-GUID ist ein HASH Wert[3] über die gespeicherte Datei. Damit kann die korrekte Einlieferung überprüft werden, denn nur wenn die Datei bei FAST vollständig und korrekt angekommen ist, kann dieser HASH Wert generiert werden und stimmt mit einem lokal generierten HASH Wert überein.

Wichtig dabei ist zu wissen, dass auf dem FAST-Speichersystem keinerlei Dateiinformation wie etwa Dateiname oder Verzeichnispfad zu einem digitalen Objekt abgespeichert wird. Jedes digitale Objekt ist also nur unter seiner Access-ID bekannt. Wenn wir die Access-ID im digitalen Katalog speichern, können wir mit dieser ID auf das Objekt zu einem spätern Zeitpunkt zurückgreifen.

Möchten wir hingegen eine Datei auf unserem File-Server vom FAST-Speichersystem zurück kopieren, weil wir sie in unserem *Workspace* vielleicht überschrieben oder gelöscht haben, wissen wir nicht, welche Access-ID die betreffende Datei gehabt hat. Es ist zwar möglich alle digitalen Objekte eines Mandanten (Archiv) vom FAST-Speichersystem zurück zu kopieren, die Daten tragen dann aber die Access-ID als Dateiname. Wir können daraus die ursprüngliche Ordnungsstruktur auf dem File-Server nicht wieder herstellen.

Das Problem wäre gelöst, würden wir nur vollständige AIPs speichern. Wir könnten dann aus der Metainformation zum AIP die entsprechende Verlinkung in den Katalog über die "Signatur" oder die richtige Platzierung in der Archivtektonik über die Metainformationen "Provenienz" und "Ordnungsstruktur" wieder herstellen. Da bei ***arcun*** aber primär davon ausgegangen wird, dass im Archiv keine AIPs verwaltet werden, sondern dass das digitale Archivgut in Form von Dateien auf einem File-Server liegt und dass die Ordnungsinformation (Tektonik) in der Pfadstruktur und/oder im Dateinamen stecken, muss es möglich sein, diese Information auch aus einem im FAST-Speichersystem gespeicherten digitalen Objekt wieder zurück zu gewinnen.

3 Kurze Skizze der Lösung

Einige Archive haben heute eine Lösung im Einsatz, welche Dateien von einem File-Server auf eine oder mehrere NAS-Boxen repliziert und die Kopien automatisch und periodisch vergleicht. Dabei ist eine selbst entwickelte Software im

---

[2] FAST bietet für das Datenspeichern ein API *(Application Programming Interface)* in C++, damit kann die Datenspeicherung in Archivsysteme wie etwa Fedora integriert werden und zusätzlich ein Programm das aus der Kommandozeile eine Datei auf die Speicherplattform übertragen kann.

[3] Hashwert *(engl. hash-code, hash-value).* Ein Hashwert ist eine möglichst eindeutige Abbildung eines grossen Datenstromes auf einen relativ kurze Zeichenkette. Eine bekannte Funktion zum Erzeugen von Hashwerten ist die MD5-Funktion. Siehe auch: http://de.wikipedia.org/wiki/Hashwert

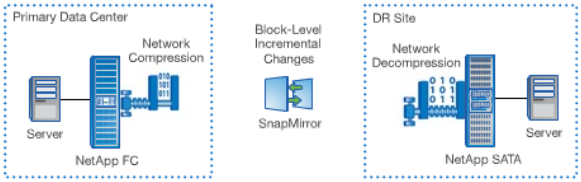
Einsatz, die ähnlich wie *NetApp SnapMirror*[4] oder *EMC Replistor*[5] funktioniert. Analog soll neben oder statt dem Kopiervorgang vom File-Server zur NAS-Box die Übertragung zum FAST Speichersystem realisiert werden.

Vor der Übertragung sollen die Dateien jeweils zusammen mit einer XMP Metadatei[6] in einen ZIP64[7] Containerfile verpackt werden. Die XMP Metadatei enthält alle aus dem Filesystem des Speicherortes automatisch extrahierbaren Informationen (Server, Verzeichnispfad, Dateiname, ctime, mtime, owner etc.)

Die vom FAST Speichersystem gelieferten Access-ID und File-GUID werden nach der Überprüfung des zurück gelieferten Hashwertes als XMP Metadaten an zwei Orten gespeichert: Erstens im ADS *(Alternate Data Stream*[8]*)* der entsprechenden Datei, wenn ein entsprechendes Dateisystem verwendet wird, oder dort am gleichen Ort als Datei *filename.xmp,* und zweitens in einem LDAP[9] oder DBM[10] Service.

---

[4] *NetApp SnapMirror Solution for Disaster Recovery:*
http://www.netapp.com/us/products/protection-software/snapmirror.html
http://media.netapp.com/documents/ds-2957.pdf



[5] *RepliStor* Daten-Recovery und -Replikation für Microsoft Windows-Umgebungen:
http://germany.emc.com/products/detail/software/replistor.htm

[6] Die Extensible Metadata Platform (XMP) ist ein Standard, um Metadaten in digitale Medien einzubetten. Der Standard wurde von Adobe im Jahr 2001 veröffentlicht. Der Standard steht mit Spezifikationen und einem SDK unter einer Open-Source-Lizenz zur Verfügung
http://de.wikipedia.org/wiki/Extensible_Metadata_Platform
http://www.adobe.com/products/xmp/indepth.html

[7] Das originale ZIP Format (Version 4.5 der Spezifikation) weist etliche Grössenbeschränkungen auf, unter anderem ist Dateigrösse auf max. 4GB beschränkt. PKWARE hat deshalb die "ZIP64" Format Erweiterung eingebracht, welche diese Begrenzungen aufhebt. Es existieren dazu bereits mehrere Implementierungen. Siehe dazu
http://en.wikipedia.org/wiki/ZIP_(file_format)

[8] In einem Dateisystem mit ADS Support oder Fork File System, z.B. NTFS oder HFS, kann eine Datei aus mehreren Streams bestehen. Streams sind Anhänge an einer Datei, die auch separat ansprechbar sind. Im Dateisystem wird aber jeweils nur der Hauptstream angezeigt, alle anderen Streams sind nicht sichtbar.
http://en.wikipedia.org/wiki/Alternate_data_stream
http://en.wikipedia.org/wiki/NTFS#Alternate_data_streams_.28ADS.29

[9] Das *Lightweight Directory Access Protocol (LDAP)*definiert einen Verzeichnisdienst, das ist eine hierarchische Datenstruktur, mit der Möglichkeit in den Knoten Schlüssel – Werte Paare zu speichern. http://de.wikipedia.org/wiki/Ldap

[10] DBM steht für einfache, dateibasierte Datenbanksysteme, siehe dazu:
http://de.wikipedia.org/wiki/DBM_(Datenbank)
Es kann natürlich jederzeit auch auf ein vollumfängliches Datenbanksystem zurückgegriffen werden.

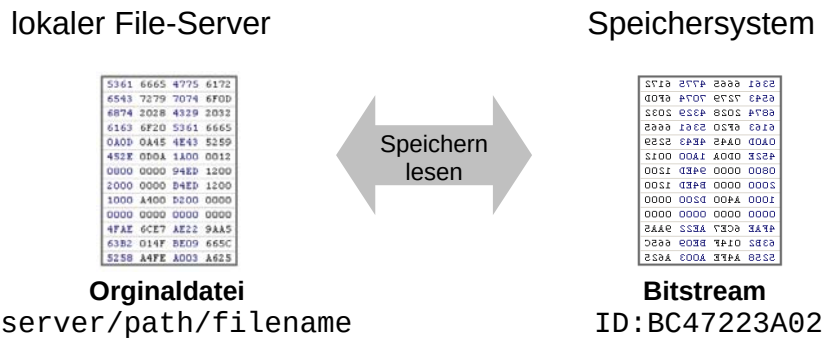
Die beteiligten Archive beabsichtigen weiterhin ein Kopie der Daten lokal im Archiv zu halten. Durch den Eintrag in ADS und LDAP / DBM ist jederzeit ersichtlich, ob und seit wann die Datei bereits auf dem FAST Speichersystem ist. Versehentlich gelöschte Dateien können über die Access-ID im LDAP oder DBM Service zurückgeholt werden. Bei Totalverlust im Archiv kann aus den XMP Metadaten in den Containerfiles die ursprüngliche Datenstruktur wieder aufgebaut werden.

Zusätzlich soll die Möglichkeit bestehen, dass Containerdateien verschlüsselt werden. Zwei Möglichkeiten stehen dafür zur Verfügung: Eine starke Verschlüsselung mit einem Schlüssel, der vom Archiv verwaltet wird, oder eine schwache Verschlüsselung mit einem pro Container neu generierten Schlüssel. Der Schlüssel wird im zweiten Fall in der XMP-Datei gespeichert und damit lokal aufbewahrt.[11]

4 Lösungsdetails

4.1 Containerdatei

Wie oben schon ausgeführt, geht bei der Speicherung auf der FAST Speicherplattform jede Metainformation, die vorher implizit in der Verzeichnisstruktur der Dateiablage enthalten gewesen ist, verloren. Eine Datei mit Dateinamen, abgelegt in einer Verzeichnishierarchie, wird zu einem Bitstream mit einem *Unique Identifier*[12] *(GUID)*. Das Problem stellt sich genau so bei andern Speicherplattformen, z.B. Amazon S3[13] oder EMC Centera. Wenn wir diese implizite Information nicht verlieren wollen, müssen wir sie zusammen mit der Datei im Speichersystem ablegen, am besten Originaldatei und Metadaten in einem Container.

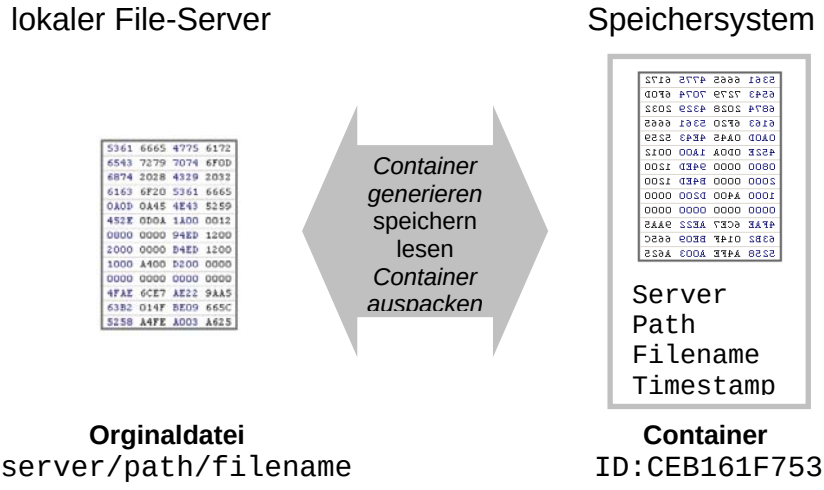


Grafik 1:Direkte Speicherung einer Datei auf dem Speichersystem, der Zugriff erfolgt über die AccessID

[11] Diese beiden Möglichkeiten der Verschlüsselung sind im Detail erläutert unter 4.5.

[12] Ein *Unique Identifier* ist ein Bezeichner, eindeutig in der Menge aller gleichen zu bezeichnenden Objekte, in dem Falle ein Textstring für alle Dateien eines Mandanten auf der FAST-Speicherplattform. Siehe dazu auch GUID *(Globally Unique Identifier)*.

[13] *Amazon Simple Storage Service:* Speicherplatz, angeboten von Amazon als Teil der Amazon Web Services, zu sehr günstigen Konditionen vor allem im Bereich E-Commerce genutzt. http://aws.amazon.com/s3/



Grafik 2:Speicherung einer Datei in einem Containerfile auf dem Speichersystem, Die Metadaten werden beim Speicher/Übertragen hinzugefügt

Am besten wird die Originaldatei vor dem Speichern völlig transparent in eine Containerdatei gepackt und mit allen aus dem Quellsystem automatisch extrahierbaren technischen Metainformationen versehen. Beim Zurücklesen wird der Container wieder geöffnet und die Originaldatei steht wieder unter ihrem ursprünglichen Dateinamen zur Verfügung oder kann wieder an den ursprünglichen Speicherort (//server/path/filename) geschrieben werden.

4.2 XMP Metadaten

Für die Speicherung der Metadaten (Servername, Verzeichnispfad, Dateiname, Zeitstempel und Datei-Besitzer und -Gruppe) kommen verschiedene Metadatenschemata in betracht. Da keines den gewünschten Fokus bietet (PREMIS und Lmer sind bei der minimalen Beschreibung zu umfangreich), wollen wir XMP als Framework und ein eigenes ***arcun***-Schema darin für unsere Metadaten wählen.

Folgende Entitäten wollen wir auf jeden Fall beim Speichern festhalten:

```
Server
Path
Filename
Timestamp (ctime, atime, mtime)
Owner / Group
Filesystem Type
```

Eine XMP Datei kann per Definition in eine Datei (Originaldatei) eingebettet werden oder als separate Datei mit der Dateiendung `.xmp` gespeichert werden. Wir wählen die letztere Lösung, weil damit die Originaldatei unverändert bleibt. Die Originaldatei und die XMP Metadatei werden zusammen in einen ZIP64 Container gepackt.

4.3 Zugriffschlüsselverwaltung

Um die Datei wieder vom Speichersystem zu holen, brauchen wir den Schlüssel, der im FAST API *AccessID* genannt wird. Wir müssen diesen Schlüssel lo-

kal verwalten, das heisst aufbewahren, damit wir jederzeit die Datei vom Speichersystem zurückholen können.

Wir können diesen Schlüssel in unserem Findmittel/Katalog (d.h. dem Archivinformationssystem) ablegen. Das setzt aber voraus, dass wir für jede Datei auch einen entsprechenden Eintrag im Findmittel haben. Weil das aber in der Regel nicht der Fall ist, können wir diese Lösung nicht weiterverfolgen.

Wenn wir weiterhin lokal eine (Arbeits-)Kopie der Datei aufbewahren, schreiben wir den Schlüssel am besten gleich zu dieser Datei. Wir können das wieder mit einer XMP Datei machen, die wir mit der Dateiendung `.xmp` zur Originaldatei speichern. Falls wir ein Dateisystem verwenden, das ADS-fähig ist (siehe oben), können wir die Information direkt in die Datei schreiben, die Originaldatei bleibt in dem Falle unverändert. Denkbar ist dabei auch, dass wir nur noch die Dateinamen mit leeren Dateien in der Verzeichnisstruktur auf unserem File-Server aufbewahren, jeweils mit der *AccessID* im ADS für den Zugriff auf die Speicherplattform.

Sinnvollerweise speichern wir die *AccessIDs* nicht nur an einem Ort, sondern pflegen im Archiv auch noch als zweite Sicherung eine kleine Datenbank mit den *AccessIDs* und dem Zugriffspfad (//server/path/filename). Es kommen dafür einfache dateibasierte Datenbanken in Fragen, zum Beispiel LDAP oder DBM: LDAP, ist ein hierarchisches Verzeichnissystem analog zum Verzeichnisbaum auf dem File-Server; DBM, zum Beispiel mit SQLite[14] realisiert, ist einfacher zu implementieren.

Wir haben also eine dreifache Sicherung, um eine oder mehrere allenfalls verloren oder gelöschte Dateien wieder herzustellen:

Wir können die *AccessID* einer Datei, die möglicherweise beschädigt oder korrupt ist, aus der XMP Metadatei bei der Datei oder im *Alternate Data Stream* der Datei lesen und die Datei zurückkopieren.

Wenn die Datei oder der ganze Verzeichnisbau verschwunden ist, können wir in unserem LDAP oder DBM System die entsprechenden *AccessID* zum fehlenden Verzeichnisbaum lesen, die Verzeichniseinträge wieder generieren und alle Dateien zurückkopieren.

Im Katastrophenfall, dass im Archiv gar keine Informationen über den File-Server mehr vorliegen und auch das DBM System gelöscht ist, können alle Dateien eines Mandanten vom FAST-Speichersystem ausgelesen und aus den Informationen im Container das ursprüngliche Dateisystem wieder aufgebaut werden.

4.4 Speicherquittung

Jede erfolgreich gespeicherte Datei wird von der FAST Speicherplattform durch zurücksenden einer Quittung, nämlich der *FileGUID* bestätigt. Wir können an-

[14] SQLite implementiert in einer Programmbibliothek ein relationales Datenbanksystem. Die Datenbank selber liegt in einer Datei, ein Datenbank Serverprozess ist nicht notwendig. www.sqlite.org/

hand der *FileGUID* feststellen, ob die Speicherung erfolgreich gewesen ist, indem wir lokal ebenfalls den Hashwert für die gespeicherte Datei errechnen und die beiden Hashwerte vergleichen. Um zu einem spätern Zeitpunkt erneut prüfen zu können, ob die Datei auf der Speicherplattform immer noch unverändert ist, müssen wir diesen Hashwert, quasi als Quittung für die Speicherung, ebenfalls aufbewahren. Sinnvollerweise machen wir das gerade zusammen mit der *AccessID* und mit den gleichen Mitteln.

4.5 Verschlüsselung

Grundsätzlich stellt sich die Frage, ob Dateien, die das Haus verlassen, nicht grundsätzlich verschlüsselt werden sollten. Sind die Dateien nicht verschlüsselt, sind wir alleine auf juristische und organisatorische Vorgaben angewiesen, was die Sicherheit vor unbefugtem Zugriff betrifft. Wenn wir die Dateien verschlüsseln, engen wir den Kreis der Leute ein auf diejenigen, welche Zugriff auf den Schlüssel haben.

Zwei Wege sind dabei möglich. Beim ersten wählen wir eine sogenannt *starke Verschlüsselung*. Der entsprechende Schlüssel kann nur mit grossem technischen Aufwand und jahrelanger Rechenzeit geknackt werden. Wir müssen den Schlüssel also im Archiv gut aufbewahren und dürfen ihn auf keinen Fall verlieren. Für die Verschlüsselung und Entschlüsselung wird der Schlüssel mit einer wiederum verschlüsselten Speicherkarte präsentiert, sonst liegt die Speicherkarte im Tresor. Geht der Schlüssel verloren, sind die Daten weg.

Die andere Möglichkeit: Wir verwenden eine sogenannt *schwache Verschlüsselung*. Die Daten lassen sich mit einigem Aufwand auch ohne Schlüssel wieder entschlüsseln. Damit es ein Angreifer nicht zu einfach hat, verschlüsseln wir jede Datei mit einem anderen Schlüssel. Damit wächst der Aufwand für das Knacken des Gesamtdatenbestandes ebenfalls ins unermessliche, aber einzelne Dateien könnten bei Bedarf immerhin zugänglich gemacht werden. Der individuelle kryptographische Schlüssel muss am Besten zusammen mit der *AccessID* und der *FileGUID* der Datei gespeichert werden.

Ein Vorteil der *schwachen Verschlüsselung* ist, dass sie mit relativ wenig Rechenaufwand bei der Ver- und Entschlüsselung im Fall der Übertragung zur Speicherplattform auskommt. Der Nachteil ist, dass der Schutz dieser Schlüssel nicht mehr so gut gewährleistet werden kann, weil wir zu jeder Datei einen eigenen Schlüssel anlegen. Die Schlüssel sind in diesem Falle, weil mit der *AccessID* zusammen abgespeichert, allen berechtigten Personen im Archiv zugänglich.