2017

LUCKYSTOR® Systembeschreibung

Thomas Firnkorn

T&T Cloud Crowding Solutions GmbH

1.1.2017

Inhalt

Überblick über LUCKYSTOR®

1 Einleitung

LUCKYSTOR® ist ein bestehendes Entwicklungs-Projekt der T&T Cloud Crowding Solutions GmbH in Zusammenarbeit mit dem Fachbereich Informatik der Hochschule Darmstadt. LUCKYSTOR® ist eine offene, plattformunabhängige und dezentrale Kommunikationstechnologie, die Nachrichten und Dateien 'Ende-zu-Ende' verschlüsselt abhörsicher überträgt und sicher speichert, um zum Schutz der Privatsphäre der Nutzer bestmöglich beizutragen. Auf Basis der LUCKYSTOR®Technologie können Firmen oder auch Privatpersonen jetzt und zukünftig einen sicheren Cloud-Speicher realisieren, der ohne zentrale Server auskommt und die Daten und Nachrichten 'schreddert' ' dann die einzelnen Schnipsel verschlüsselt und die einzelnen Schnipsel auf mehrere ggf. weltweit verteilte Knoten ablegt. Somit hat kein Administrator Zugriff auf alle Dateiteile und selbst wenn ein Verschlüsslungsalgorithmus kompromittiert werden würde, hat er keine Möglichkeit die Datei zu lesen. Das Ziel, „ No Single Point of Attack" wurde erreicht.

Mit dieser Cyber Secure Cloud Solution (CSCS) kann eine sichere Plattform zukünftig für industrielle Anwendungen im Internet of Things bereitgestellt werden. LUCKYSTOR verbindet erstmals beide Welten, die IKT Branche und die Industrie, Handel und Dienste Branche zum sicheren Internet of Things.

Neben der LUCKYSTOR®-Knoten Software sind folgende LUCKYSTOR® Clients verfügbar:

- LUCKYSTOR® ftp - sicherer File-Transfer
- LUCKYSTOR® sync - sichere Dateisynchronisation auf mehreren Endgeräten
  LUCKYSTOR® sync2 - sichere Dateisynchronisation auf Industriellen Endgeräten
- LUCKYSTOR® share - sicheres Teilen von Dokumenten
- LUCKYSTOR® fs - sicheres Dateisystem für Mac und Linux
- LUCKYSTOR® web - Web-Oberfläche (Firefox, Safari, IE, ...), die die verschiedene Clients integriert und sichere Chat-Funktionalität anbietet
- LUCKYSTOR® im - sicherer Messenger für IOS und Android.

Damit ist etwa folgendes Anwendungsszenario umsetzbar:

Ein Konzern hat mehrere eigenständige Auslandseinheiten mit jeweils eigenen Rechenzentren. Täglich können die unternehmenskritischen Daten jeder Einheit sicher mittels LUCKYSTOR®ftp und LUCKYSTOR® sync über die Auslandseinheiten als Backup verteilt werden, ohne dass ein Administrator einer Einheit Zugriff auf die Daten einer anderen Auslandseinheit hat. LUCKYSTOR®fs ermöglich es, dass jede Einheit auf die eigenen Backups in der Cloud mit Filebrowsern (Windows, Mac und Linux/Unix) wie auf die eigenen Festplatten zugreifen kann. Durch LUCKYSTOR®share können Einheiten auch Daten sicher teilen.

Durch die Verwendung des erweiterten Chord-Verfahrens sind hochskalierbare LUCKYSTOR®-Netze möglich, die z.B. im Umfeld des Internet of Things und Industrie 4.0 eingesetzt werden können, um eine sehr große Anzahl von Clients effektiv bedienen zu können.

2 Architektur

Die verschiedenen Anwendungen, die auf LUCKYSTOR® basieren, sind historisch als getrennte Anwendungen gewachsen und wurden später zusammengeführt. In diesem Kapitel soll ein Überblick gegeben werden über die verschiedenen vorhandenen Anwendungen und wie sie zusammenhängen. Die Geräte die an LUCKYSTOR® beteiligt sind können in zwei Kategorien aufgeteilt werden, so wie in Abbildung 2 dargestellt. Die Server sind LUCKYSTOR®-Knoten, auf denen die Clients Blöcke speichern können. Ein Rechner kann Server und Client gleichzeitig sein, wie es in peer-to-peer Netzen üblich ist, je nachdem welche Software auf dem Gerät ausgeführt wird. Das Grundgerüst bilden die Server, die LUCKYSTOR®-Knoten, auf denen Datenblöcke in Form von Key-Value-Paaren abgelegt werden können. Sie sind in einem Ring angeordnet und mithilfe

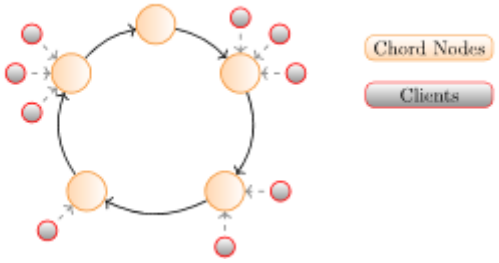
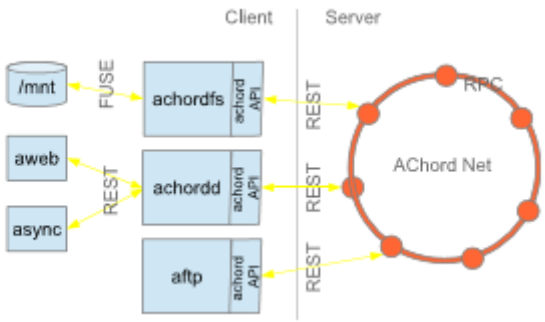


Abbildung 1: Clients und LUCKYSTOR® P2P Netz als Cloud Speicher

des am Massachusetts Institute of Technology (MIT) entwickelten Chord-Algorithmus, erweitert um spezielle Sicherheitsaspekte können Inhalte effizient gefunden werden. Die Software die auf ihnen läuft heißt LUCKYSTOR® nchord. Geräte die diese Software ausführen werden im Folgenden als Server bezeichnet. Die Abbildung 2 gibt einen Überblick über das Zusammenspiel der verschiedenen Software-Komponenten von LUCKYSTOR®.

Abbildung 2: Das Zusammenspiel der LUCKYSTOR®-Komponenten:



Der Nutzer interagiert mit einem der Commandlinetools, dem Web Interface oder mit dem nativen Dateisystem seines Rechners. Diese kommunizieren mit dem LUCKYSTOR®-Netzwerk direkt oder über den LUCKYSTOR®-daemon. Alle Komponenten auf der linken Seite des Trennstrichs sind lokal auf der Maschine des Nutzers (Client) ausgeführt. Mithilfe der LUCKYSTOR®-API kommunizieren sie über eine REST-Schnittstelle mit der Serversoftware LUCKYSTOR®nchord. Die nchord-Knoten verwenden einen RPC-Mechanismus zum effizienten Austausch von Verwaltungsinformationen.

Die LUCKYSTOR® ftp-library beschreibt ein Application Programming Interface (LUCKYSTOR®-API) für einfache dateisystemähnliche Schnittstellen zur Nutzung des Blockspeicher an. Dabei werden Dateisystemaufrufe auf get- und put-Operationen umgesetzt. Für alle das System verlassenden Blöcke gilt, dass sie zuvor verschlüsselt und mit einem Message Authentication Code (MAC) versehen

werden. Außerdem gibt es eine gleichnamige LUCKYSTOR®ftp-Anwendung für die Kommandozeile, mit der Dateien in das LUCKYSTOR®-Netzwerk hoch- und heruntergeladen werden können. Zusätzlich kann man in ftp-Manier Dateien von einem Rechner auf den anderen übertragen, wobei das LUCKYSTOR® Peer to Peer Netzwerk als Transportmittel verwendet wird.

Der LUCKYSTOR®-daemon ist eine Software die im Hintergrund auf Clients läuft um Anfragen der LUCKYSTOR®ftp-Anwendung, von LUCKYSTOR®sync und dem web interface für LUCKYSTOR® zu bedienen. Der LUCKYSTOR®-daemon bediente ursprünglich nur async, wurde dann um Schnittstellen für das web interface erweitert und die Funktionen der LUCKYSTOR®ftp-Anwendung wurden dorthin migriert.

Async ist eine Anwendung zur Dateisynchronisation über mehrere Endgeräte des gleichen Nutzers. Async ist ein Kommandozeilenprogramm. Es kommuniziert mit dem LUCKYSTOR®-daemon über eine REST-Schnittstelle. Die gesamte Funktionalität um Befehle des Nutzers auszuführen und regelmäßige Synchronisationsvorgänge zu starten ist im LUCKYSTOR®-daemon implementiert.

LUCKYSTOR® Ashare ist ein Projekt bei dem das Teilen von Dateien mit und ohne Synchronisation umgesetzt wurde.

LUCKYSTOR®fs ist ein verteiltes Dateisystem für Unix mit dessen Hilfe Dateien und Verzeichnisse die ein Nutzer ins Chord-Netzwerk gelegt hat im Dateibrowser des jeweiligen Betriebssystems angezeigt werden können. Mithilfe von FUSE ist LUCKYSTOR®fs geeignet für den Dateisystembedarf gängiger Unix-Programme. Das web interface ist eine lokal ausgeführte Website, die ihre Inhalte über den LUCKYSTOR®-daemon per REST-Schnittstelle lädt. Über das web interface können async, ashare und aftp bedient werden. Für die Umsetzung aller Komponenten außer des web interfaces wurde die von Google entwickelte Programmiersprache Go gewählt, die besonders komfortable Werkzeuge zur Serverentwicklung und Entwicklung von nebenläufigen Programmen bietet.

3 Syncronisation

Um die Latenzzeiten in IP Netzen bei großen Entfernungen zu eliminieren wird die Chord Technologie um die Funktionalität des Etna/Paxos Verfahrens ergänzt. Etna /Paxos basiert auf DHT Distributed Hash Table und ermöglicht die Nutzung von verteilten Dateisystemen in Echtzeit. Damit wird sichergestellt, dass gespeicherte Daten in verteilten Speichern konsistent sind. Der Einsatz von unterschiedlichen Cloud Diensten in einem logischen LUCKYSTOR Ring ist damit möglich.

4 Verschlüsslung

Die in LUCKYSTOR® verwendete Krypto-Technologie basiert auf der NaCl Bibliothek des Krypto-Experten Daniel J. Bernstein. Das nachfolgende Diagramm stellt den Vorgang im Überblick dar.

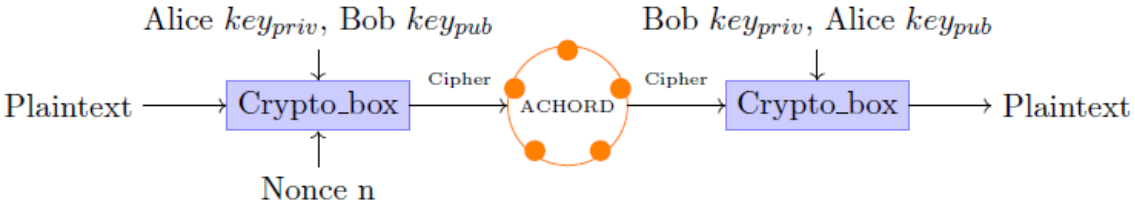


Abbildung 3: LUCKYSTOR® Krypto Systematik

Die Crypto box ist der Kern der NaCl-Bibliothek und benötigt den 'Plaintext' (zu verschlüsselnde Nachricht oder Dateiblock), den PublicKey des Empfängers, den PrivateKey des Senders, sowie einer Nonce (Number used Once) zur Verschlüsselung mittels asymmetrischem 256 Bit ECDH (Elliptic Curve Diffie-Hellman). Zunächst wird mit Hilfe des 32 Byte PublicKey des Empfängers (Bob) ein 32 Byte (256 Bit) symmetrischer Schlüssel ksym abgeleitet, der anschließend zur Verschlüsselung des Plaintext mittels des symmetrischen Salsa20 verwendet wird. Bei Salsa20 handelt es sich um eine Streamcipher, der vier Mal performanter ist als der normalerweise verwendeten AES Algorithmus.

Dieser symmetrische Schlüssel wird für jede Nachricht bzw. jeden Block einer Datei neu berechnet und unterscheidet sich durch die 24 Byte lange Nonce immer von den vorherigen Versionen, somit wird ein 'Mitleser' stets unterschiedliche verschlüsselte Informationen desselben Textes erheblich erschwert. Nach der Verschlüsselung wird der Cipher gebildet aus mit ksym verschlüsseltem Plaintext, asymmetrisch verschlüsselten ksym und Nonce. Die einzelnen so verschlüsselten Nachrichtenteile bzw. Blöcke einer Datei werden auf dem LUCKYSTOR®-Ring verteilt und stehen zum Abruf bereit. Beim Empfänger(Bob) wird der Cipher über die crypto box- Funktionen analog der Verschlüsslungssystematik entschlüsselt, also aus dem Cipher werden die Bestandteile extrahiert, dann wird der verschlüsselte ksym mit seinem eigenen geheimen Schlüssel entschlüsselt und damit die verschlüsselte Nachricht bzw. der Dateiblock entschlüsselt und man erhält den Plaintext.

Das o.a. Verfahren wird in allen LUCKYSTOR®-Clients verwendet, um die Ende-zu-Ende Verschlüsselung zu implementieren. D.h. sowohl in aftp zum sicheren Senden von Dateien von Alice zu Bob, als auch zum Speichern von Dateien in der LUCKYSTOR®-Cloud.

5 Installation und Aufruf der Komponenten

Die Installation und Verwendung der LUCKYSTOR®-Programme wird am Beispiel des u.a. LUCKYSTOR®Rings demonstriert.

Abbildung 4: LUCKYSTOR®-Ring mit IP:Port Information

4.1 Voraussetzungen

Die Client- und Knotenprogramm sind in der Programmiersprache ’Go’ entwickelt. Ausführbare Programme können für die in Tabelle 1 aufgelisteten Plattformen erzeugt werden. Die Leistungsfähigkeit des Cloud-Speichers wird im Wesentlichen von der Güte des Netzes zwischen den LUCKYSTOR®-Knoten und der Netz-Anbindung der Client bestimmt. Die LUCKYSTOR®-Knoten sollten auf Servern mit mindestens 8GB Hauptspeicher und schnellen Festplatten installiert

OS	Architektur
darwin	386
darwin	amd64
darwin	arm
darwin	arm64
dragonfly	amd64
freebsd	386
freebsd	amd64
freebsd	arm
linux	386
linux	amd64
linux	arm
linux	arm64
linux	ppc64
linux	ppc64le
netbsd	386
netbsd	amd64
netbsd	arm
openbsd	386
openbsd	amd64
openbsd	arm
plan9	386
plan9	amd64
solaris	amd64
windows	386
windows	amd64

Tabelle 1: Betriebssystem und Rechnerarchitektur

werden, eine Minimalanforderung gibt es jedoch nicht. Die exakten Anforderungen bzgl. Hauptspeicher, Festplattengröße und Knotenanzahl ergeben sich aus dem Einsatzzweck (viele kleinere Daten oder wenige große Daten pro Filetransfer). Der LUCKYSTOR® Cloud-Speicher eignet sich sowohl für große Datenmengen als auch für sehr viele Transfers.

5.2 Knotensoftware LUCKYSTOR® nchord

Die Knotensoftware besteht lediglich aus dem Programm ’nchord’. Es ist ein für die jeweilige Plattform kompilierte ausführbare Datei, die wie folgt aufgerufen wird (hier am Beispiel eines Linux Servers).

Zunächst der Aufruf für den ersten LUCKYSTOR®-Knoten mit der Adresse IP1:

```
1 $ nchord -rpc=IP1:4001 -http=IP1:8001 -dir=/AHOME/data &
```

Hiermit wird der RPC-Port für die Kommunikation der Knoten untereinander und der http-Port für den Zugriff der Clients sowie das Verzeichnis für die Daten definiert.

Nun die Aufrufe der restlichen LUCKYSTOR®-Knoten am Beispiel der Adresse IP3:

```
1 $ nchord -rpc=IP3:4001 -http=IP3:8001 -join=IP1:4001 -dir=/AHOME/data &
```

Mittels der Option ’-join’ wird einer der bereits laufenden nchord-Knoten angegeben.

Usage of nchord:

-M migrate blocks on shutdown

-alsologtostderr

log to standard error as well as files

-compat

registers compatibility endpoints (default true)

-debug

registers a http handler at /debug/node/ that dumps a node's state as JSON (default true)

-dfrags number

number of fragments sufficient for reconstruction of erasure-coded blocks (default 1)

-dir directory

use given directory for persistent storage

-efrags number

number of fragments created for erasure-coded blocks (default 3)

-exp

enables experimental features (might cause nasal demons)

-grpc addr

service Etna RPCs on addr (default same interface as -rpc, random port)

-http addr

serve http on addr (default ":8000")

-log_backtrace_at value

when logging hits line file:N, emit a stack trace

-log_dir string

If non-empty, write log files in this directory

-logflush duration

flush logs in intervals of duration (e.g. 1s or 2m30s)

-logtostderr

log to standard error instead of files

-nreplica number

number of replicas in an Etna configuration (default 3)

-pushca string

use these PEM-encoded certs to verify trust chain

-pushcert file

load certificate to present to push service from this file

-pushkey file

file with private key for pushcert

-pushmaxretry n

retry push server notification n times (default 3)

-pushurl string

notify about incoming IM messages on this URL

-rpc addr

service chord RPCs on addr (default ":4000")

-stderrthreshold value

logs at or above this threshold go to stderr

-trace

trace Etna execution

-v value

log level for V logs

-version

print version and exit

-vmodule value

comma-separated list of pattern=N settings for file-filtered logging

5.3 Client-Programme

Um die Client-Programme verwenden zu können, muss zunächst einmalig ein Schlüsselpaar für den Benutzer erstellt werden. Dazu dient das Programm LUCKYSTOR®ftp-keygen.

```
$ aftp -keygen

$ ls -al ~/.achord

$-rw------- 1 user staff 173B 10 Jun 12:41 id 4 -rw------- 1 user staff 53B 10 Jun 12:41 id.pub

$
```

Im Verzeichnis $HOME/.LUCKYSTOR® sind die beiden Schlüssel, der private und der öffentliche Schlüssel als Dateien id bzw. id.pub abgelegt.

Achtung

Der Verlust des Schlüsselpaares bedeutet den Verlust aller in der LUCKYSTOR®-Cloud gespeicherten Daten! Es ist absolut erforderlich, ein Backup dieser beiden Dateien auf einem externen Medium zu erstellen bzw. eine ausgeprägte Schlüsselverwaltungssystematik einführen.

Der Aufruf der Client-Programme wird am Beispiel 'LUCKYSTOR®ftp' zum Up- und Download beschrieben.

```
$ ls -l h*
 ls: h*: No such file or directory
 $
 $ aftp -c IP1:8001 aput /etc/hosts 2>/dev/null
 $ aftp -c IP2:8001 als 2>/dev/null
 hosts
 $ aftp -c IP5:8001 aget hosts 2>/dev/null
 $ ls -l h*
 -rw------- 1 as staff 595 11 Aug 13:07 hosts
```

Die Client-Programme geben über die Option '-c' an, wo sich ein LUCKYSTOR®-Knoten befindet.

6 Schlüsselverwaltung

Alle LUCKYSTOR-Clients verwenden eine Ende-zu-Ende Verschlüsslung bei dem einen Benutzer ein Paar von privatem und öffentlichen Schlüssel auf dem eigenen Rechner, oder einem USB-Stick / Keycard abgelegt hat. Der Verlust dieses Schlüsselpaars führt dazu, dass seine in der LUCKYSTOR-Cloud gespeicherten Dateien nicht mehr zugreifbar sind - eine Katastrophe für den Benutzer. Im Folgenden
wird ein System zur Schlüsselverwaltung beschrieben, dass es einem Benutzer erlaubt, ein Backup seines Schlüsselpaars sicher in einem LUCKYSTOR-Netz eines Dienstanbieters anzulegen, ohne dass ein Dienstanbieter die Schlüssel des Benutzers lesen kann. Somit kann der Benutzer im vorher beschriebenen Katastrophenfall sein Schlüsselpaar aus dem LUCKYSTOR-Netz des Dienstanbieters wiederherstellen und somit wieder auf seine Cloud-Daten zugreifen. Das Verfahren ist geeignet, beliebige Schlüsselpaare sicher in einer LUCKYSTOR-Cloud zu verwahren, ist also nicht beschränkt auf Schlüsselpaare von LUCKYSTOR-Client.

Die Verwaltung der Schlüssel wird anhand des folgenden Szenarios erklärt:

Alice verwendet einen AChord Client um ihre Daten sicher in einer AChord Cloud zu speichern. Sie hat also ein Schl•usselpaar auf ihrem Rechner, das sie gegen Verlust, etwa durch einen Plattencrash ihres Rechners schützen will.

Abbildung 5: AChord Cloud die Alice verwendet um ihre Dateien zu speichern

Bob stellt einen Schlüssel Backup Service bereit, bei dem die Schlüssel der Kunden in seinem eigenen LUCKYSTOR-Netz sicher abgelegt werden.

Abbildung 6: Bob's AChord Cloud um seinen Backup Service anbieten zu können

Bob verwendet das Programm 'almgr', um für Kunden Lizenzen zu erzeugen, die sie brauchen, um mittels des von Bob zur Verfügung gestellten Programms 'akmgr' die Verwaltung ihrer Schlüssel vorzunehmen. Bob erzeugt eine Lizenz für Alice. Dabei verwendet er das Programm 'almgr': Die erzeugte Lizenz '4521-76cd-65af-e6a2-fec0' sendet Bob an Alice.
Mittels des einfachen Programms kann Bob Lizenzen verwalten, neue erstellen und bereits erstellte auflisten. Es ist die Kundendatenbank von Bob.
Alice hat von Bob eine Lizenz erhalten und kann damit das Programm zur Verwaltung ihrer Schlüssel aufrufen. Beim Aufruf gibt sie ihre Mail-Adresse, den Lizenzstring und die Adresse von Bob's AChord-Cloud an. Nun kann Sie ihre Schlüsselpaare auflisten, den aktuellen Schlüssel zum Zugriff auf ihre AChord-Cloud •andern oder neue Schlüsselpaare erzeugen. Oder aber ein Backup von ihren Schlüsselpaare machen bzw. ein Restore ihrer Schlüsselpaare durchführen.
Mittels 'akmr' kann Alice beliebig viele Schlüsselpaare erzeugen, im Beispiel oben angedeutet einen zur Ablage der privaten Daten und einen für die geschäftlichen Daten. Somit kann sie ihren "Namensraum" für AChord-Dateien partitionieren.

6.1. Verschlüsselung durch Passworte

Aus den o.a. Beispielen erkennt man, dass das Programm 'akmgr' eine Schlüsselsystematik mittels Passworten verwendet. Dazu existiert ein AChord-Package, orientiert an Kyle Isom's Buch, das im Rahmen der Entwicklung des LUCKYSTOR SDK entstanden ist.
Zum Ver- und Entschlüsseln wird das mit GO DevOps mitgelieferte Package NaCl (Networking and Cryptography library) des Crypto-Experten Daniel J. Bernstein verwendet. NaCl beinhaltet die symetrische Bibliothek 'secretbox' und die asymetrische Bibliothek 'box'. Secretbox verwendet den Stream Cipher 'XSalsa20' zur Bewahrung der Vertraulichkeit und als MAC 'Poly1305'.
Hier wird ein einfacher Mechanismus zum Schlüsselaustausch verwendet, nämlich ein Passwort zum Berechnen des Crypto-Schlüssels. Verwendung findet das Paket 'golang.org/x/crypto/scrypt'. Dabei kann man die Güte des Crypto-Schlüssels über Parameter, wie Hauptspeicher und CPU und Anzahl der Iterationen zum Erzeugen des sichen Crypto-Schlüsssels bestimmen. In den Programmen wird dies als Variable 'quality' hinterlegt. Es muss eine 2er Potenz sein. Sinnvolle Werte für die Anzahl der Iterationen beim Hashen sind:

- minimale Sicherheit 2^2 = 4
- normale Sicherheit 2^{14} = 16384
- hohe Sicherheit 2^{20} = 1048576

In der Anwendung 'akmgr' wird die höchste o.a. Stufe des 'qualitty-Parameter' verwendet.

'Encrypt' verschlüsselt die Nachricht mittels eines Passwortes und eines Güteparameters. Dabei wird zunächst ein zufälliger Salt-Wert ermittelt, dann aus dem Passwort, dem Salt und dem Güteparameter ein Crypto-Schlüssel berechnet. Anschließend wird die Nachricht mit dem Crypto-Schl•ussel verschlüsselt und der Salt-Wert vorangestellt. Somit erhält man insgesamt immer unterschiedliche Resultate, auch wenn dasselbe Passwort auf einen Plaintext angewendet werden.

Die Funktion 'Decrpyt' trennt Salt und verschlüsselten Text, berechnet aus dem Passwort und dem Güteparameter den Crypto-Schlüssel, der dann verwendet wird um den Text zu entschlüsseln.
Zur Verschlüsselung der Schlüsselpaare wird ein tar-File erzeugt, das in Blöcke fester Größe zerteilt wird. Jeder Block wird mittels 'Encrypt' unter Verwendung des von Alice angegebenem Passwort verschlüsselt und in Bob's AChord-Cloud hochgeladen. Alle Block-Adressen (Hashes der Block-Inhalte) werden auf Bob's AChord-Netz verteilt sowie eine 'Index-Datei' mit den Block-Adressen des tar-Archives. Zur Adressierung der Index-Datei von Alice wird neben ihrer Mail-Adresse, die Lizenz und ihr Password verwendet. Zur Entschlüsselung der zum tar-Archiv gehörenden Blöcke wird die o.a. Funktion 'Decrypt' verwendet.
Insgesamt kann nun Alice ihre Schlüsselpaare sicher in Bob's AChord-Cloud sichern. Nur Alice kennt ihr Passwort, das sie braucht, um die gespeicherten Schlüsselpaare wiederherstellen zu können, um im Katastrophenfall keine ihrer Cloud-Daten zu verlieren.

Literaturhinweise

Ion Stoica u.a.; Chord: a scalable peer-to-peer lookup protocol for internet applications; ACM SIGCOMM Computer Communication Review 31.4 (2001)

Kyle Isom; Practical Cryptography With Go; https://leanpub.com/gocrypto/read