

Diplomarbeit

sblit

ausgeführt an der
Höheren Abteilung für Informationstechnologie/Netzwerktechnik
der Höheren Technischen Lehranstalt Wien 3 Rennweg

im Schuljahr 2014/2015

durch

**Martin Exner
Andreas Novak
Nikola Szucsich**

unter der Anleitung von

August Hörndl

Wien, 21. April 2015

Kurzfassung

Diese Diplomarbeit zielt darauf ab, einen Dienst zur Synchronisation von Dateien über das Internet in der Form freier Software zu schaffen. Zentrale Stellen wie Server sollen dabei im Sinne der Sicherheit vor Eingriffen durch Unbefugte vermieden werden. Statt der Zwischenspeicherung von Daten auf Cloudspeicher eines herkömmlichen Anbieters werden Daten in verschlüsselten Blöcken auf den Geräten anderer Nutzer zwischengespeichert.

An die Stelle einer klassischen Client-Server-Infrastruktur tritt ein dezentrales Peer-to-Peer-Netzwerk. Dadurch und durch die Verteilung von Daten auf Geräte anderer Teilnehmer auftretende Problemstellungen sollen im Rahmen dieser Diplomarbeit gelöst werden. Dazu zählen dezentrale Adressierung, dezentrales Routing, Fairness von gegenseitiger Speicherfreigabe, Sicherheit und Effizienz von sowohl Kommunikation als auch Synchronisation sowie Behandlung von Synchronisationskonflikten.

Abstract

This diploma project aims at creating free software for file synchronization over the internet. Central elements like servers should be avoided in order to prevent unauthorized individuals from accessing the system. Instead of storing files in data centers, which is the case with conventional cloud providers, data is split into small, encrypted blocks and then stored on other users' devices.

While other file synchronization services implement a classic client-server infrastructure, this project utilizes a decentralized peer-to-peer network for communication. Problems that emerge due to the distribution of data in small, encrypted blocks and due to the use of a decentralized peer-to-peer network should be solved in this thesis. Those problems include decentralized addressing, decentralized routing, fairness of mutual memory allocation, security as well as efficiency of both communication and synchronization and solving synchronization conflicts.

Ehrenwörtliche Erklärung

Ich versichere,

- dass ich meinen Anteil an dieser Diplomarbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe
- und mich auch sonst keiner unerlaubten Hilfe bzw. Hilfsmittel bedient habe.

Wien, am 21. April 2015

Martin Exner

Andreas Novak

Nikola Szucsich

Präambel

Die Inhalte dieser Diplomarbeit entsprechen den Qualitätsnormen für „Ingenieurprojekte“ gemäß § 29 der Verordnung des Bundesministers für Unterricht und kulturelle Angelegenheiten über die Reife- und Diplomprüfung in den berufsbildenden höheren Schulen, BGBl. Nr. 847/1992, in der Fassung der Verordnungen BGBl. Nr. 269/1993, Nr. 467/1996 und BGBl. II Nr. 123/97.

Liste der betreuenden Lehrer:

Prof. DI August Hörndl, *Hauptbetreuer*

Prof. DI Franz Breunig, *Betreuer*

Prof. DI Herbert Saschofer, *Betreuer*

Inhaltsverzeichnis

1. Überblick	1
1.1. Ausgangssituation	1
1.1.1. Einleitung	1
1.1.2. Funktionsweise üblicher Dateisynchronisationsdienste	1
1.2. Problematik	2
1.3. Lösung	3
1.3.1. Einleitung	3
1.3.2. Herausforderungen	3
1.3.3. Szenarien	6
2. Decentralized Communication Layer	9
2.1. Einleitung	9
2.2. Anforderungen an die Kommunikationsschicht	9
2.3. Network Types	10
2.3.1. Notwendigkeit	10
2.3.2. Definition	10
2.3.3. Notation	10
2.3.4. Circle Network	10
2.4. Adressierung	11
2.4.1. Notwendigkeit	11
2.4.2. Adressierung mit RSA	11
2.4.3. Überprüfung von Adressen	11
2.4.4. Eindeutigkeit von Adressen	12
2.4.5. Adressformat	12
2.5. Routing	14
2.5.1. Notwendigkeit	14
2.5.2. Schwierigkeit	14
2.5.3. Distanzbasiertes Routing	14
2.5.4. Umsetzung im Quellcode von DCL	15
2.6. NAT-Traversal	17
2.6.1. Einleitung	17
2.6.2. Probleme für Peer-to-Peer-Verbindungen	18
2.6.3. NAT Hole Punching	19
2.6.4. Einschränkungen	19
2.7. Links	20
2.7.1. Einleitung	20
2.7.2. Sicherheit	20
2.7.3. Channels	20
2.7.4. Basic Management Channel Protocol	21

2.8.	Interservice-Protokoll	25
2.8.1.	Einleitung	25
2.8.2.	Address Slots	25
2.8.3.	Network Slots	27
2.8.4.	Connection Base	27
2.8.5.	Initialisierung	28
2.8.6.	Messageaufbau	28
2.8.7.	Messages	29
2.9.	Application-to-Service-Protokoll	35
2.9.1.	Einleitung	35
2.9.2.	Sicherheit	37
2.9.3.	Network Endpoint Slots	37
2.9.4.	Remote Keys	37
2.9.5.	Initialisierung	38
2.9.6.	Messageaufbau	38
2.9.7.	Messages	38
2.10.	Application Channels	48
2.10.1.	Einleitung	48
2.10.2.	Sicherheit	48
2.10.3.	Aufbau	49
2.10.4.	CRISP	50
2.11.	Packet Components	50
3. sblit		55
3.1.	Einleitung	55
3.2.	Konfiguration	55
3.2.1.	Allgemein	55
3.2.2.	freceivers.txt	55
3.2.3.	logs.txt	56
3.2.4.	receivers.txt	58
3.2.5.	rk.txt	58
3.2.6.	sblitDirectory.txt	58
3.2.7.	symmetricKey.txt	58
3.2.8.	uk.txt	58
3.3.	Datei-Verarbeitung	59
3.3.1.	Allgemein	59
3.3.2.	Versionierung	59
3.3.3.	Löschen auf Partnergeräten	59
3.3.4.	Reaktionen auf Dateiänderungen	59
3.3.5.	Konflikte	63
3.4.	Kommunikation	66
3.4.1.	Allgemein	66
3.4.2.	Nachrichten an alle Geräte	66
3.4.3.	Nachrichten an eigene Geräte	67
3.4.4.	Nachrichten an Partnergeräte	72
3.4.5.	Ablauf	77

Inhaltsverzeichnis

3.5. Partnerschaften	79
3.5.1. Allgemein	79
3.5.2. Partnergeräte	79
3.5.3. Delta-Daten	79
3.5.4. Dauer einer Partnerschaft	79
3.5.5. Sicherheit	79
3.5.6. Wunschpartnerschaften	80
3.5.7. Beispiel	80
4. Graphical User Interface	83
4.1. Einleitung	83
4.2. System-Tray	83
4.3. Überblicksfenster	84
4.4. Konfigurationsmenü	85
4.5. Konfigurationsmenü	86
4.5.1. Einführung	86
4.5.2. Vorteile	86
4.5.3. Nachteile	86
4.5.4. Aufbau	86
5. Website	89
5.1. Allgemein	89
5.2. Diplomarbeitswebsite	89
5.3. Produktwebsite	89
6. Wettbewerbe	91
6.1. Allgemein	91
6.2. Wettbewerbe	91
6.2.1. Jugend Innovativ	91
6.2.2. u19 – Create your World	92
6.2.3. AXAWARD	92
6.2.4. Internet of Things Cup	92
6.2.5. Computer Creative Wettbewerb	93
A. Anhang 1	95
B. Anhang 2	101
C. Anhang 3	123
Akronyme	127
Glossar	129

Tabellenverzeichnis

2.1. FlexNum – Darstellbare Werte	52
---	----

Abbildungsverzeichnis

1.1. Üblich gehandhabtes Uploaden einer Datei	2
1.2. Üblich gehandhabtes Downloaden einer Datei	3
1.3. Heimrechner ist nicht erreichbar. Die Datei kann nicht synchronisiert werden.	4
1.4. Heimrechner ist wieder erreichbar, sodass die Datei nun vom Server bezogen werden kann.	5
1.5. Übertragung einer Datei zwischen zwei erreichbaren Hosts.	6
1.6. Hochladen einer Datei in die dezentrale Filecloud.	7
1.7. Herunterladen einer Datei aus der dezentralen Filecloud beziehungsweise von den Partnergeräten.	8
 2.1. Link Packet	20
2.2. BMCP – Genereller Messageaufbau	21
2.3. BMCP – Connect Request Message	22
2.4. BMCP – Connect Reply Message	22
2.5. BMCP – Disconnect Message	23
2.6. BMCP – Kill Message	23
2.7. BMCP – Crypto Init Message	23
2.8. BMCP – Ack Message	24
2.9. BMCP – Change Protocol Request Message	24
2.10. BMCP – Channel Block Status Request Message	24
2.11. BMCP – Channel Block Status Report Message	26
2.12. BMCP – Open Channel Request Message	27
2.13. BMCP – Throttle Message	27
2.14. Interservice-Protokoll – Genereller Messageaufbau	29
2.15. Interservice-Protokoll – Version Message	29
2.16. Interservice-Protokoll – LLA Request Message	29
2.17. Interservice-Protokoll – LLA Reply Message	30
2.18. Interservice-Protokoll – Trusted Switch Message	31
2.19. Interservice-Protokoll – Crypto Challenge Request Message	31
2.20. Interservice-Protokoll – Crypto Challenge Reply Message	32
2.21. Interservice-Protokoll – Connection Base Notice Message	32
2.22. Interservice-Protokoll – Network Join Notice Message	33
2.23. Interservice-Protokoll – Network Leave Notice Message	34
2.24. Interservice-Protokoll – Integration Request Message	34
2.25. Interservice-Protokoll – Integration Connect Request Message	34
2.26. Interservice-Protokoll – Network Packet Message	35
2.27. Interservice-Protokoll – Application Channel Slot Assign Message	36
2.28. Interservice-Protokoll – Application Channel Data Message	36
2.29. Application-to-Service-Protokoll – Genereller Messageaufbau	38

2.30. Application-to-Service-Protokoll – Revision Message	38
2.31. Application-to-Service-Protokoll – Generate Key Message	39
2.32. Application-to-Service-Protokoll – Join Network Message	39
2.33. Application-to-Service-Protokoll – Slot Assign Message	40
2.34. Application-to-Service-Protokoll – Data Message	41
2.35. Application-to-Service-Protokoll – Address Public Key Message	41
2.36. Application-to-Service-Protokoll – Join Default Networks Message	41
2.37. Application-to-Service-Protokoll – Key Encrypt Message	42
2.38. Application-to-Service-Protokoll – Key Decrypt Message	42
2.39. Application-to-Service-Protokoll – Key Crypto Response Message	43
2.40. Application-to-Service-Protokoll – Application Channel Outgoing Request Message	44
2.41. Application-to-Service-Protokoll – Application Channel Incoming Request Message	45
2.42. Application-to-Service-Protokoll – Application Channel Accept Message . .	46
2.43. Application-to-Service-Protokoll – Application Channel Connected Message	47
2.44. Application-to-Service-Protokoll – Application Channel Data Message . .	47
2.45. Application-to-Service-Protokoll – Key Encryption Block Size Request Message	48
2.46. Application-to-Service-Protokoll – Key Number Response Message	48
2.47. CRISP – Genereller Messageaufbau	50
2.48. CRISP – Neighbor Request Message	51
2.49. KeyComponent – Genereller Componentaufbau	53
2.50. KeyComponent – RSA Key Component	54
3.1. sblit – Authenticity Request Message	67
3.2. sblit – Authenticity Response Message	67
3.3. sblit – File Request Message	68
3.4. sblit – File Response Message	69
3.5. sblit – File Message	70
3.6. sblit – File Delete Message	71
3.7. sblit – Device Refresh Message	72
3.8. sblit – Partner File Request Message	73
3.9. sblit – Partner File Response Message	73
3.10. sblit – Partner File Message	75
3.11. sblit – Partner File Delete Message	76
3.12. sblit – File Delete Partner Message	76
3.13. Wahl eines Geräts, auf dem Speicherplatz freigegeben wird	80
3.14. Austausch und Eintragen der Adressen	81
3.15. Eintragen der Geräte des Partners	81
3.16. Eintragen der Adressen des Partnergeräts	81
4.1. Menüfenster beim Systemtray	83
4.2. Überblicksfenster	84
4.3. Konfigurationsmenü	85
4.4. Beispiel-GUI zur Erklärung	88
C.1. Vorderseite der Visitenkarte	123

Abbildungsverzeichnis

C.2. Rückseite der Visitenkarte	123
---	-----

1. Überblick

1.1. Ausgangssituation

1.1.1. Einleitung

Zahlreiche Dienste wie Dropbox, Google Drive, iCloud, Microsoft Onedrive und Box bieten ihren Kunden kostenlosen Cloudspeicher an. Der Vorteil bei der Verwendung solcher Dateisynchronisationsdienste liegt in der Einfachheit, mit der diese Dienste Datenbestände auf mehreren Geräten synchron halten.

Besonders bei Verwendung eines Laptops in der Schule und eines zweiten Rechners Zuhause ist die Anmeldung bei einem dieser Anbieter hilfreich. Statt Dateien mühsam über externe Speichermedien wie USB-Sticks kopieren zu müssen, kann ein Synchronisationsclient des Anbieters installiert werden, der neue und geänderte Dateien von einem Gerät automatisch auf den Cloudspeicher lädt und somit auf das zweite Gerät synchronisiert.

Die meisten Cloudspeicher sind so aufgebaut, dass die Daten der Nutzer zentral und unverschlüsselt auf Servern des Anbieters gespeichert werden. Dieses Modell ist zwar effizient und vergleichsweise einfach umzusetzen, birgt aber Risiken für die Privatsphäre der Nutzer.

Durch die zentrale Speicherung der Daten entsteht ein einziger Angriffspunkt, der, wenn ein Angriff erfolgreich oder ein Fremdzugriff auf einem anderen Weg möglich ist, den Zugang zum gesamten System und damit zu den Daten aller Nutzer erlaubt. Der Verzicht auf die Verschlüsselung der Daten ermöglicht es einem Angreifer außerdem, die Daten vollständig auszuwerten.

sblit verfolgt statt einer klassischen Client-Server-Infrastruktur einen vollständig dezentralen Ansatz in Kombination mit starker Verschlüsselung, wodurch es keinen zentralen Angriffspunkt mehr gibt, der die Privatsphäre der Nutzer gefährdet.

1.1.2. Funktionsweise üblicher Dateisynchronisationsdienste

1.1.2.1. Allgemein

Die Erläuterung der Funktionsweise üblicher Dateisynchronisationsdienste soll beim Verstehen der grundsätzlichen Idee von sblit helfen, da diese auf den sicherheitsbezogenen Problemen basiert, die mit der Speicherung von Daten auf einer zentralen Stelle einhergehen.

Übliche Dateisynchronisationsdienste bauen auf einer Client-Server-Struktur auf. Hierbei werden Dateien auf den zentralen Servern des betreibenden Unternehmens übertragen, zwischengespeichert, verwaltet und schließlich synchronisiert.

Als Erklärung des üblichen Funktionsablauf eines Synchronisationsvorgangs von Dateien folgt ein Szenario.

1.1.2.2. Szenario

In der Schule wird ein Übungsprotokoll geschrieben. Wird die Übung nicht fertiggestellt, muss sie zu Hause fertiggestellt werden.

Beim Erstellen und Bearbeiten des Übungsprotokolls auf dem Schulrechner wird eine Kopie dieser Datei auf den vom Betreiber zur Verfügung gestellten Cloudspeicher beziehungsweise dessen Server hochgeladen und dort gespeichert. Der Sender behält dabei die Originaldatei. (Abbildung 1.1.2.2, Szenario)

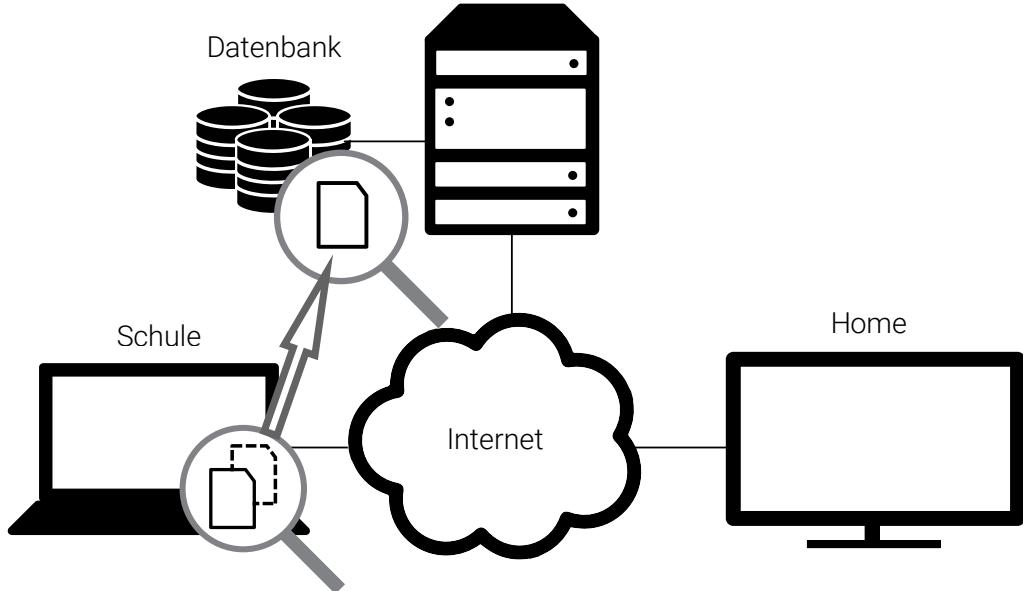


Abbildung 1.1.: Üblich gehandhabtes Uploaden einer Datei

Der Schulrechner wird heruntergefahren, zu Hause wird der Heimrechner hochgefahren. Die Synchronisations-Software wird sobald gestartet, über die Änderungen im Cloudspeicher informiert und aktualisiert den lokalen Datenbestand, in diesem Fall durch Herunterladen des neu angelegten Übungsprotokolls vom Server. (Abbildung 1.1.2.2, Szenario)

Das Übungsprotokoll hat nach dem Synchronisationsvorgang auf dem Heim-PC den gleichen Inhalt wie die Datei auf dem Schulrechner, da es sich um die selbe Version handelt. Würde das Übungsprotokoll nun auf dem Heim-PC bearbeitet werden, würde die neue Version analog dazu zuerst auf den Server geladen werden und vom Server aus auf die anderen Rechner synchronisiert werden.

1.2. Problematik

Einen Server als Dreh- und Angelpunkt eines Dienstes einzusetzen scheint die am einfachsten umzusetzende und kosteneffizienteste Methode zu sein, birgt aber Sicherheitsrisiken für den Benutzer.

In der Vergangenheit kam es schon öfters zu Hackerangriffen auf große Firmen, bei denen große Mengen an Kundendaten gestohlen wurden. Vorfälle wie diese zeigen, wie leicht Mengen an privaten Daten in falsche Hände fallen können, wenn diese zentral an einem Ort gespeichert werden.

1. Überblick

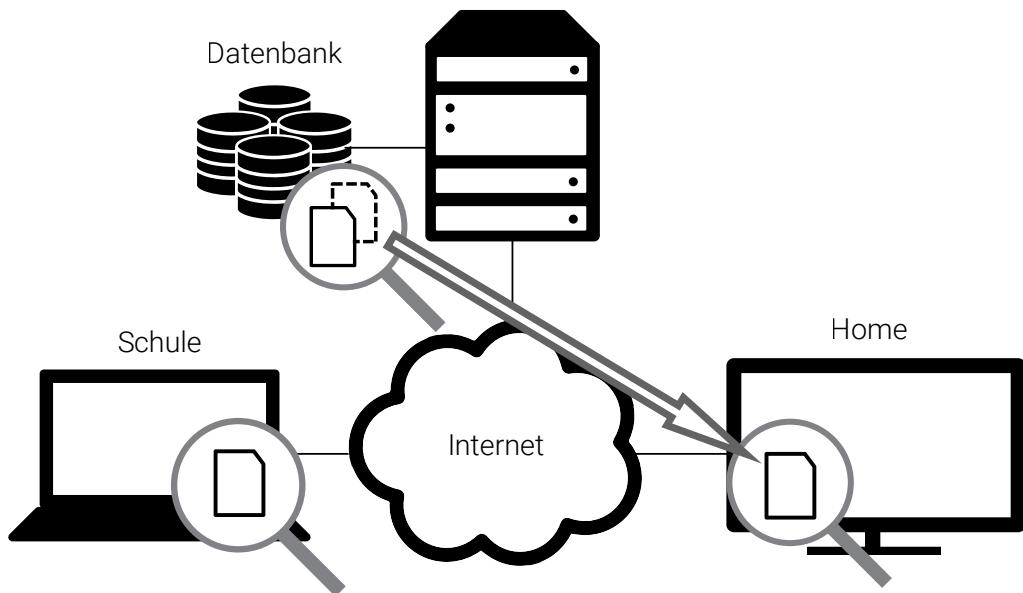


Abbildung 1.2.: Üblich gehandhabtes Downloaden einer Datei

Die Speicherung privater Daten von Usern auf zentralen Servern eines Unternehmens ebnet Geheimdiensten und staatlichen Sicherheitsbehörden außerdem den Weg, flächendeckende Überwachung durchzuführen und damit massiv in die Privatsphäre aller User des Synchronisationsdienstes einzudringen, da das Unternehmen als Anbieter solcher Cloudspeicher unter Umständen zur Herausgabe der Nutzerdaten gesetzlich verpflichtet ist.

Hauptursache für diese Problematik ist die unverschlüsselte und vor allem zentrale Speicherung der Daten.

Bei sblit werden die angesprochenen Probleme umgangen, ohne den durch Synchronisationsdienste gebotenen Komfort zu mindern.

1.3. Lösung

1.3.1. Einleitung

Anders als bei herkömmlichen Dateisynchronisationsanwendungen, findet die Datenübertragung bei sblit nicht über zentrale Server des Betreibers statt, sondern über eine direkte, verschlüsselte Verbindung zwischen den beteiligten Geräten. Durch diesen Ansatz fällt die zentrale Sammelstelle von Nutzerdaten weg und dem Datendiebstahl von Hackern wird ein Riegel vorgeschoben. Außerdem wird kein betreibendes Unternehmen mehr benötigt, um die Server zur Verfügung zu stellen, wodurch auch die einheitliche Anlaufstelle für staatliche Sicherheitsbehörden und Geheimdienste wegfällt.

1.3.2. Herausforderungen

1.3.2.1. Allgemein

Durch die Dezentralisierung entstehen aber auch neue Herausforderungen, die bei herkömmlichen Dateisynchronisationsdiensten durch den Einsatz zentraler Stellen vergleichsweise

einfach bewältigt werden können, welche in den folgenden Abschnitten erläutert werden.

1.3.2.2. Verbindung von zu synchronisierenden Geräten

Bei sblit fehlt der öffentliche Server als bekannte Vermittlungsstelle in der Kommunikation zwischen den zu synchronisierenden Geräten, sodass sich diese über einen speziellen Mechanismus finden müssen, um eine direkte Verbindung aufzubauen zu können. Das Kapitel 2 Decentralized Communication Layer, Seite 9 widmet sich der Bewältigung dieser, sowie weiterer durch die Dezentralisierung der Kommunikation entstehenden Herausforderungen.

1.3.2.3. Alternative zu Server als Filecloud

Der Server agiert aber nicht nur als Vermittlung der Clients zueinander, damit diese kommunizieren können, sondern auch als Zwischenspeicher für Daten. Dieser ist nämlich essenziell, um Dateien auch dann synchronisieren zu können, wenn stets nur eines der zu synchronisierenden Geräte gleichzeitig eingeschaltet ist. Um zu erklären warum das so ist, folgt ein Szenario mit Verwendung eines Servers: Der Rechner in der Schule ist eingeschaltet, das Gerät zu Hause nicht. Wird nun eine bereits synchronisierte Datei auf dem Schulrechner bearbeitet, wird eine Kopie dieser wie üblich auf die Filecloud, also den Server hochgeladen. Wenn der Unterricht vorbei ist, wird der Schulrechner abgedreht und zu Hause nach Einschalten des Heim-PCs kann sofort die neueste Version der geänderten weiterbearbeitet werden, obwohl der Schulrechner ausgeschaltet ist, da die Datei in der Filecloud zwischengespeichert und von dort aus auf den Heimrechner übertragen wurde.

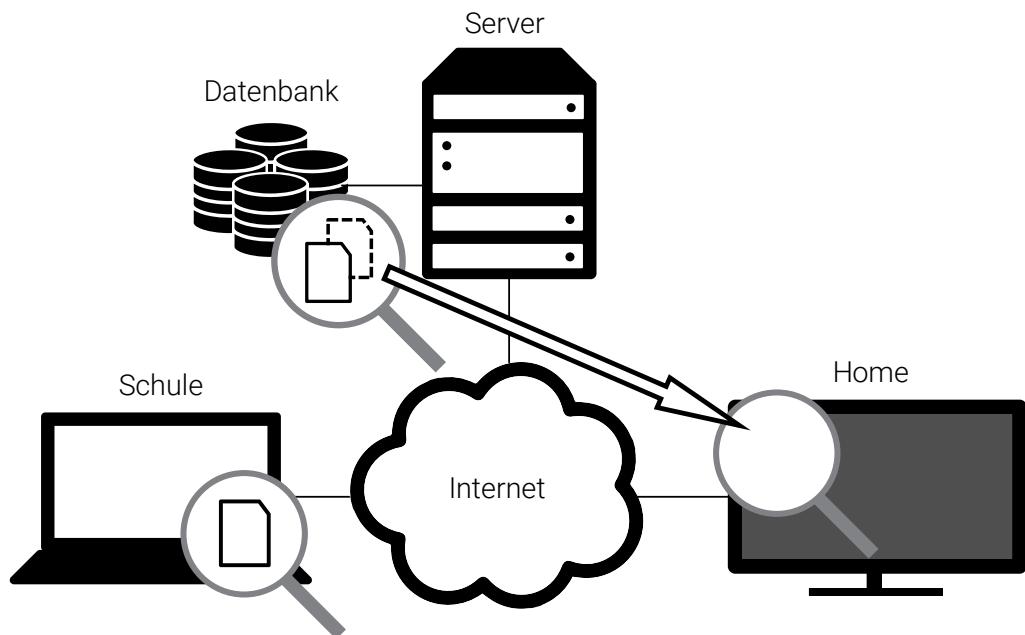


Abbildung 1.3.: Heimrechner ist nicht erreichbar. Die Datei kann nicht synchronisiert werden.

Ohne den Server als Filecloud wäre die alte Version der Datei bearbeitet worden, sodass zwei neue Versionen der Datei existieren würden. Es wäre also ein Synchronisationskonflikt

1. Überblick

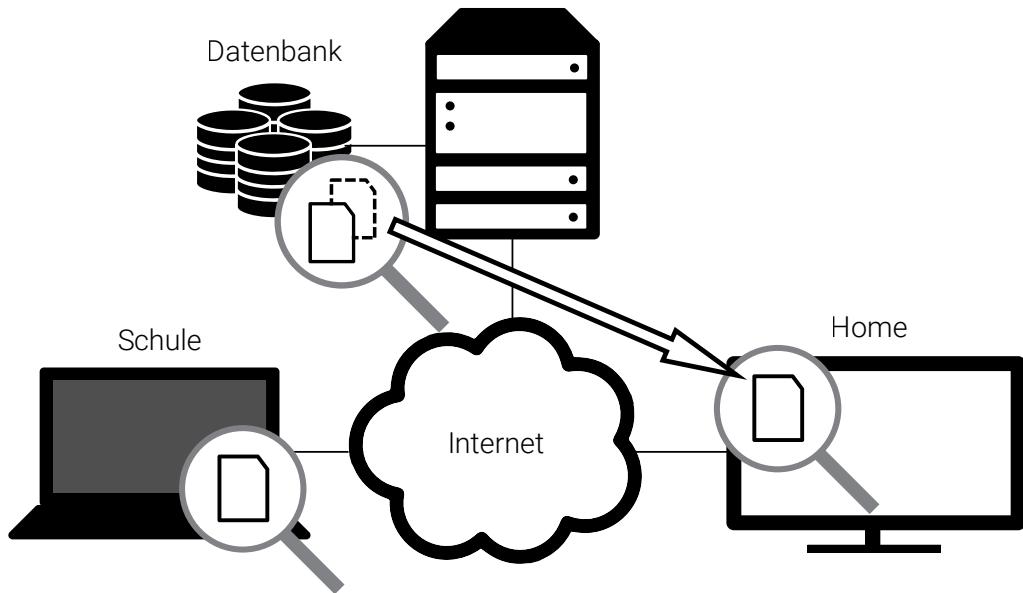


Abbildung 1.4.: Heimrechner ist wieder erreichbar, sodass die Datei nun vom Server bezogen werden kann.

aufgetreten. Durch den Verzicht auf den Server muss also eine alternative Lösung gefunden werden, um Dateien zwischenspeichern und Synchronisationskonflikte vermeiden zu können.

Bei sblit kommt deshalb eine dezentrale Filecloud zum Einsatz. Anstelle der zentralen Speicherung auf einem Server wird zum Zwischenspeichern der Daten eines Nutzers auf den Speicherplatz auf den Geräten anderer Nutzer von sblit zurückgegriffen. Die Geräte der Nutzer speichern sich also gegenseitig die Änderungen ihrer Datenbestände in verlüsselter Form.

Dieses Prinzip funktioniert durch ein Peer-to-Peer-Netzwerk, welches durch faire Abkommen zwischen den sblit-Nutzern, sogenannte Partnerschaften, realisiert wird. Grundsätzlich sind Partnerschaften einfach eine Vereinbarung mit anderen Nutzern, um sich gegenseitig Speicherplatz freizugeben. Eine Partnerschaft wird entweder automatisch mit den Geräten unbekannter Nutzer geschlossen oder durch den Nutzer selbst mit den Geräten bekannter. Das Kapitel 3.5 Partnerschaften, Seite 79 widmet sich dieser Thematik genauer.

Beim externen Zwischenspeichern auf Partnergeräten sollen diese allerdings nie vollständigen Zugriff auf fremde Dateien erhalten, welche bei ihnen gespeichert wurden. Deshalb werden extern zu speichernde Dateien in viele Datenblöcke aufgeteilt. Diese Blöcke werden verschlüsselt und verstreut auf den Partnergeräten gespeichert. Die Information zum korrekten Zusammensetzen der Datei besitzen dabei nur der Ersteller selbst und dessen Synchronisationspartner.

Für die Besitzer der Partnergeräte ist es somit unmöglich auf die ursprüngliche Datei zu schließen. Sollte es nämlich einer von ihnen schaffen die Datenblöcke zu entschlüsseln, würde ihnen noch immer der Rest an Datenblöcken fehlen und die Information zum korrekten Zusammensetzen dieser.

Von einer ständigen Erreichbarkeit darf auch bei den Partnergeräten nicht ausgegangen werden. Problematisch wäre es, wenn die Datei nicht von der dezentralen Filecloud

heruntergeladen werden kann, nur weil eines der Partnergeräte nicht erreichbar ist. Um das zu vermeiden, wird jeder verschlüsselte Datenblock mehrmals in der Filecloud gespeichert, sodass nur ein Bruchteil der Partnergeräte erreichbar sein müssen um die komplette Datei anfordern zu können.

Um den Ablauf eines Synchronisationsvorgangs bei sblit vollständig aufzuzeigen, folgen nun häufig auftretende Szenarien.

1.3.3. Szenarien

1.3.3.1. Allgemein

Bei sblit wird der Inhalt eines Ordners synchronisiert, der bei der Erstinstallation der Anwendung angegebenen wurde. Sobald sich Dateien innerhalb des Ordners ändern, wird die neue Version der Datei kopiert und die Kopie wird den erreichbaren Synchronisationspartnern über eine direkte verschlüsselte Verbindung gesendet.

1.3.3.2. Synchronisieren bei erreichbaren Synchronisationspartnern

Wenn auf dem Schulrechner eine neue Datei erstellt oder bereits synchronisierte verändert und gespeichert wird, dann wird diese Datei in mehrere Datenblöcke aufgeteilt, diese Blöcke werden verschlüsselt und direkt über eine verschlüsselte Verbindung an den Heimrechner gesendet, sofern der jeweilige Synchronisationspartner erreichbar ist. Beim Synchronisationspartner angekommen, werden diese entschlüsselt und zu der vollständigen neuen Datei zusammengesetzt.

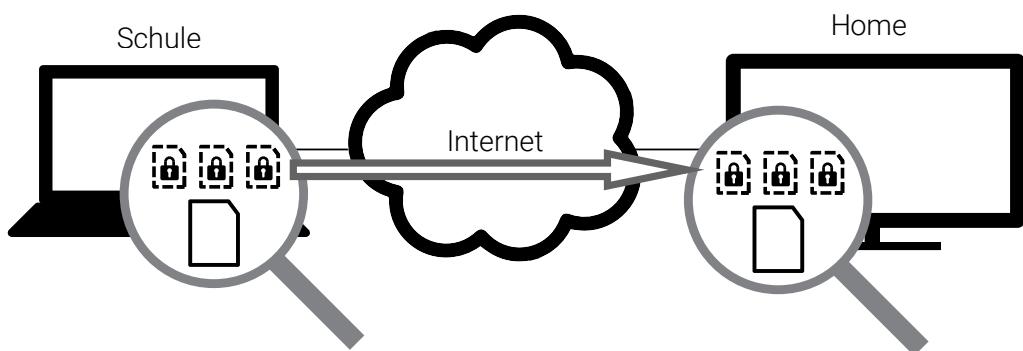


Abbildung 1.5.: Übertragung einer Datei zwischen zwei erreichbaren Hosts.

1.3.3.3. Synchronisieren bei nicht erreichbaren Synchronisationspartnern

Für Synchronisationspartner, die nicht erreichbar sind, wird die Datei in der dezentralen Filecloud zwischengespeichert. Die Datei wird kopiert und in Blöcke aufgeteilt. Diese Blöcke werden verschlüsselt und verteilt auf den Partnergeräten gespeichert.

Da die Erreichbarkeit der gesamten Datei auf den Partnergeräten zu gewährleisten ist, wird die Datei mehrmals in die dezentrale Filecloud gespeichert, sodass nur ein Bruchteil der Partnergeräte erreichbar sein muss, um auf die vollständige Datei zugreifen zu können.

1. Überblick

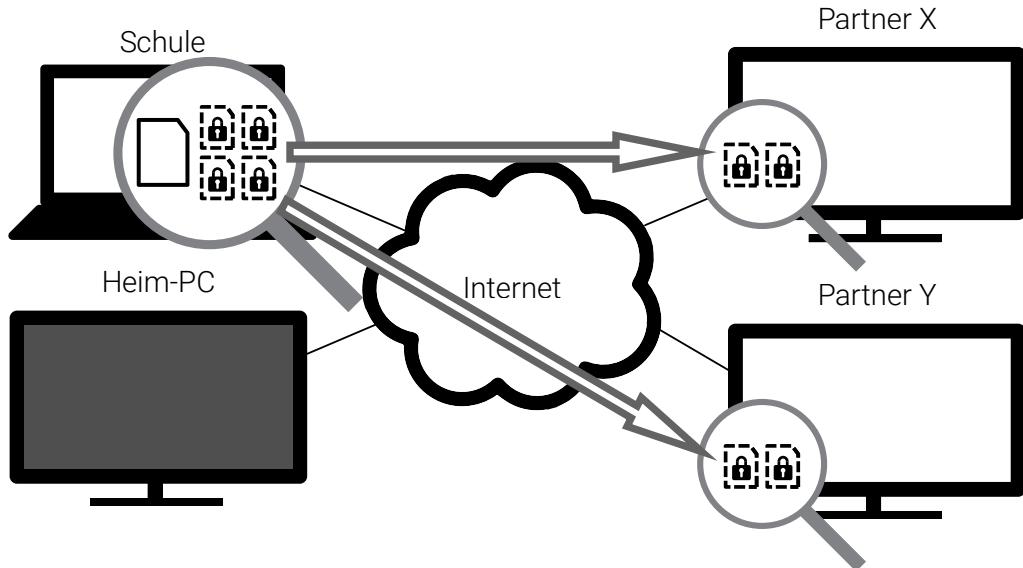


Abbildung 1.6.: Hochladen einer Datei in die dezentrale Filecloud.

Sobald der Synchronisationspartner, hier der Heim-PC, wieder hochgefahren ist, fordert er die verschlüsselten Dateiblöcke von den Partnergeräten an. Die Blöcke werden über direkte verschlüsselte Verbindungen gesendet, entschlüsselt und zu der vollständigen neuen Datei zusammengesetzt.

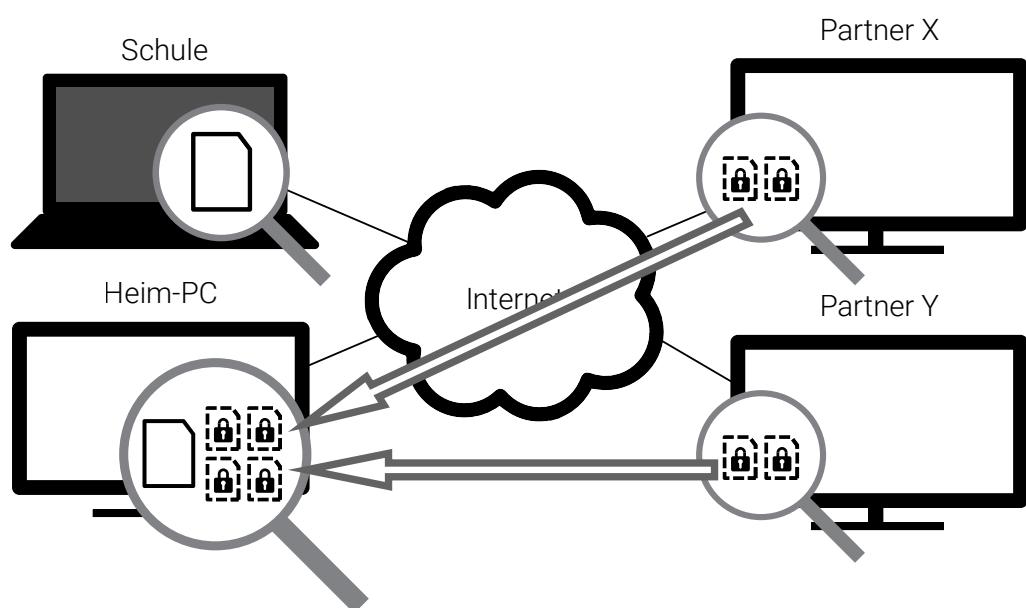


Abbildung 1.7.: Herunterladen einer Datei aus der dezentralen Filecloud beziehungsweise von den Partnergeräten.

2. Decentralized Communication Layer

2.1. Einleitung

Um die Hauptanforderung an die Umsetzung, den Verzicht auf zentrale Server im System, realisieren zu können, wird ein Peer-to-Peer-Netzwerk benötigt. Über dieses läuft die Kommunikation der Synchronisationsanwendung. Dabei muss das Netzwerk vollständig dezentral aufgebaut sein, um ohne Server funktionieren zu können. Gleichzeitig müssen alle Teilnehmer des Netzwerks die selben Berechtigungen haben, weswegen es keinen Teilnehmer geben kann, der besondere Befugnisse hat. Deshalb ist es notwendig, dass sich das Netzwerk und die Teilnehmer selbst organisieren und verwalten, ohne dabei auf ein zentrales Organ angewiesen zu sein.

Dieses Peer-to-Peer-Netzwerk ist in einer separaten Kommunikationsschicht, dem Decentralized Communication Layer (DCL), umgesetzt. Das ermöglicht es einerseits, DCL für Anwendungen von fremden Entwicklern zu öffnen, sodass diese auf das schon bestehende Netzwerk zurückgreifen können und nicht erst ein eigenes umsetzen müssen, und andererseits erleichtert die klare Abgrenzung der Funktionen zwischen Kommunikationsschicht und eigentlicher Synchronisationsanwendung die Umsetzung beider erheblich.

2.2. Anforderungen an die Kommunikationsschicht

Das Peer-to-Peer-Netzwerk des DCL muss eine Reihe von Eigenschaften aufweisen, um für die Anwendung eingesetzt werden zu können. Diese Eigenschaften decken sich im wesentlichen mit üblichen Anforderungen an herkömmliche, nicht dezentrale Netzwerke und lauten wie folgt:

Addressierung

Den Teilnehmern müssen eindeutige Adressen zugewiesen werden können, die auf ihre Echtheit überprüfbar sind.

Routing

Zwischen Teilnehmern müssen anhand ihrer Adressen Kommunikationskanäle aufgebaut werden können.

Durch den dezentralen Ansatz von DCL gestaltet sich die Realisierung dieser Eigenschaften jedoch anders als das beispielsweise für ein übliches Computernetzwerk oder das Internet der Fall wäre. So können Adressen nicht von zentralen, dazu bemächtigten Behörden vergeben werden und das Routing kann nicht von speziellen Teilnehmern des Netzwerks in einer hierarchischen Organisation erfolgen.

2.3. Network Types

2.3.1. Notwendigkeit

Um DCL für eine breite Menge an Anwendungen benutzbar zu machen, muss es möglich sein, alle Eigenschaften, die ein Peer-to-Peer-Netzwerk zur Kommunikation zwischen den Instanzen dieser Anwendungen haben muss, erfüllen zu können. Das lässt sich am besten realisieren, indem der DCL so umgesetzt wird, dass mehrere unterschiedliche Peer-to-Peer-Netzwerke darüber aufgebaut werden können. Das hat den Vorteil, auch mit Peer-to-Peer-Netzwerken, die andere Anforderungen als bisher existente Netzwerke haben, auf die bereits bestehende Menge an Hosts aus dem DCL zurückgreifen zu können. So können neue Netzwerke schnell und stabil zur Verfügung gestellt werden und neue Teilnehmer erhöhen nicht nur die Verfügbarkeit ihres Netzwerks, sondern auch die aller anderer auf DCL aufbauender Netzwerke.

2.3.2. Definition

Ein Network Type beschreibt ein Netzwerk innerhalb des DCL durch Festlegung folgender Eigenschaften:

Adresskonzept

Beinhaltet die Länge von Adressen und die Verfahren zur Bildung sowie zur Überprüfung von Adressen.

Routingverfahren

Beinhaltet den Vorgang zur Weiterleitung von Nachrichten, die an andere Teilnehmer des Netzwerks adressiert sind.

2.3.3. Notation

Für jeden Network Type gibt es einen sogenannten Network Type Identifier, mit dem er identifiziert wird. Dieser setzt sich aus einem Typstring im Stil von Java Package Names und einem Attributstring für Untereigenschaften zusammen.

2.3.4. Circle Network

Das im DCL standardmäßig verwendete Netzwerk ist das Circle Network. Der Typstring des Network Type des Circle Network ist `org.dclayer.circle`.

Adressen werden im Circle Network standardmäßig durch Anwendung des Hashalgorithmus SHA-1 auf die öffentlichen RSA-Schlüssel erzeugt und haben eine Länge von 20 Bytes, was der Digest Length von SHA-1 entspricht.

Das Format des Attributstrings des Network Type des Circle Network ist `Hashalgorithmus/Adresslänge`. Der standardmäßige Attributstring für den Network Type des Circle Network ist also `sha1/20`.

Daraus ergibt sich der standardmäßige Network Type Identifier des Circle Network, `org.dclayer.circle sha1/20`.

2.4. Adressierung

2.4.1. Notwendigkeit

Um in einer auf dem DCL basierenden Anwendung Synchronisationgruppen bilden zu können, ist es nötig, die Teilnehmer dieser Gruppen als solche erkennen zu können. Das erfordert wiederum die permanente und eindeutige Adressierung dieser Teilnehmer. Da sich die öffentlichen IP-Adressen der meisten privaten Internetanschlüsse und somit des Großteils der Zielgruppe von sblit periodisch ändern, eignen sich diese jedoch nicht als Adressen im DCL. Es wird daher ein anderes Adresskonzept benötigt, mit dem ohne zentrale Stelle allen Teilnehmern eindeutige Adressen zugewiesen werden können, die auf ihre Echtheit überprüfbar sind.

Es ist nicht ausreichend, die Teilnehmer ihre Adressen willkürlich selbst bestimmen zu lassen und eine Funktion zu implementieren, die überprüft, ob eine neu generierte Adresse im Netzwerk schon existiert, da dieser Ansatz keinerlei Sicherheit vor einer absichtlichen Übernahme der Adresse eines anderen Teilnehmers durch einen Angreifer bietet.

Dieses Unterkapitel beschreibt die Adressierung im Circle Network.

2.4.2. Adressierung mit RSA

Asymmetrische Verschlüsselungsverfahren wie RSA eignen sich durch ihre Eigenschaften ausgezeichnet für die Adressierung innerhalb eines Netzwerks, in dem alle Teilnehmer die gleichen Berechtigungen haben und in dem keine höhere Instanz existiert, die Adressen vergeben und diese verifizieren kann.

Bei asymmetrischen Verschlüsselungsverfahren kommen sogenannte Schlüsselpaare, bestehend aus zwei Schlüsseln, zum Einsatz. Die Besonderheit liegt darin, dass eine Nachricht, die mit einem Schlüssel aus dem Schlüsselpaar verschlüsselt wurde, bei asymmetrischen Verfahren im Gegensatz zu symmetrischen Verfahren nicht mit dem selben Schlüssel auch wieder entschlüsselt werden kann, sondern ausschließlich mit dem anderen Schlüssel des Schlüsselpaares. Gleichzeitig kann aus einem Schlüssel eines Schlüsselpaares der dazugehörige andere Schlüssel des Schlüsselpaares nicht in absehbarer Zeit berechnet werden.

Dadurch wird es möglich, ein Adressierungssystem umzusetzen, das die Anforderungen im Bezug auf Überprüfbarkeit der Adressen erfüllt. Dazu wird eine Adresse angenommen, die sich von einem der beiden Schlüssel aus dem Schlüsselpaar ableitet. Dieser Schlüssel wird bewusst veröffentlicht, während der andere Schlüssel aus dem Schlüsselpaar geheim gehalten wird. Der Schlüssel aus dem Schlüsselpaar, der veröffentlicht wird, wird auch *Öffentlicher Schlüssel* oder *Public Key* genannt, der weiterhin geheim gehaltene Schlüssel *Privater Schlüssel* oder *Private Key*.

Öffentliche Schlüssel als Grundlage für Adressen haben den Vorteil, dass die Adressen ohne höhere Behörde oder zentrale Stelle auf ihre Echtheit überprüft werden können und somit fälschungssicher sind. Ein Mechanismus zur Überprüfung solch einer Adresse wird im nächsten Abschnitt beschrieben.

2.4.3. Überprüfung von Adressen

Dadurch, dass eine mit einem öffentlichen Schlüssel verschlüsselte Nachricht nicht wieder mit dem öffentlichen Schlüssel entschlüsselt werden kann, sondern ausschließlich mit dem dazugehörigen privaten Schlüssel, kann der Besitz des gesamten Schlüsselpaares bewiesen

werden, ohne mehr als den öffentlichen Schlüssel preisgeben zu müssen: Eine beliebige Folge von Daten wird vom überprüfenden Teilnehmer generiert, mit der Grundlage der Adresse des zu überprüfenden Teilnehmers, also seinem öffentlichen Schlüssel verschlüsselt und anschließend an diesen übermittelt. Dort wird die empfangene Nachricht vom zu überprüfenden Teilnehmer mit seinem privaten Schlüssel, den nur dieser Teilnehmer besitzt, wieder entschlüsselt und zurück an den überprüfenden Teilnehmer gesendet.

Decken sich die ursprünglich vom überprüfenden Teilnehmer generierten Daten mit denen, die vom zu überprüfenden Teilnehmer entschlüsselt wurden, ist der Besitz des gesamten Schlüsselpaares, und nicht lediglich des öffentlichen Schlüssels, bewiesen. In anderen Worten, die vom überprüften Teilnehmer bekanntgegebene Adresse ist echt.

2.4.4. Eindeutigkeit von Adressen

Die Eindeutigkeit der generierten RSA-Schlüsselpaare und somit der Adressen kann zwar nicht garantiert werden, eine Kollision ist jedoch aufgrund der Länge der verwendeten Schlüssel und der Anzahl der dadurch möglichen Schlüsselpaare dermaßen unwahrscheinlich, dass davon ausgegangen werden kann, dass eine Kollision praktisch nicht auftritt. [2]

2.4.5. Adressformat

2.4.5.1. RSA-Schlüssellänge

Die für die Adressen im DCL verwendeten öffentlichen RSA-Schlüssel haben eine Länge von 2048 Bits, was 256 Bytes entspricht. IPv4- und IPv6-Adressen haben mit jeweils 32 Bits (4 Bytes) bzw. 128 Bits (16 Bytes) vergleichsweise kurze Adressen. Trotzdem ist eine Adressierung mit 128 Bits mehr als ausreichend und wesentlich längere Adressen, wie sie beispielsweise bei direkter Verwendung von RSA-Schlüsseln entstehen, bringen lediglich Nachteile und keinerlei Vorteile mit sich, da der dadurch entstehende Overhead bei der Übertragung von Nachrichten eine wesentliche Reduktion der Effizienz zur Folge hat.

Gleichzeitig ist die Schlüssellänge mit 2048 Bits aber bereits im unteren Bereich dessen angesiedelt, was zur Zeit noch als sicher eingestuft wird. [3] Die Verwendung kürzerer RSA-Schlüssel würde die Sicherheit der Adressen im DCL und der über den DCL laufenden Kommunikation negativ beeinflussen.

Der folgende Abschnitt beschreibt, wie im Circle Network des DCL die öffentlichen RSA-Schlüssel so verarbeitet werden, dass die resultierenden Adressen die Effizienz der Kommunikation nicht mindern.

2.4.5.2. Adressverkürzung

Die im DCL verwendeten Adressen decken sich nicht mit den öffentlichen RSA-Schlüsseln, sondern basieren lediglich darauf. Da ohne Weiterverarbeitung der Schlüssel zu lange Adressen entstünden, wird auf die Daten der öffentlichen Schlüssel ein Hashalgorithmus angewandt, welcher die Daten in einem ersten Schritt wesentlich verkürzt. In einem optionalen zweiten Schritt können die bereits verkürzten Daten weiter verkürzt werden, um unnötig lange Adressen zu eliminieren.

Nachfolgend ist jener Teil des Quellcodes von DCL angeführt, welcher im `CircleNetworkType` zur Adressverkürzung angewandt wird.

2. Decentralized Communication Layer

```
1 Data scaleAddress(Address address) {  
2  
3     Data fullData = address.toData();  
4     Data hashedData = hash.update(fullData).finish();  
5  
6     if(byteLength == hashedData.length()) {  
7         return hashedData;  
8     }  
9  
10    Data scaledData = new Data(byteLength);  
11  
12    for(int i = 0; i < hashedData.length(); i++) {  
13        int scaledIndex = i % scaledData.length();  
14        scaledData.setByte(scaledIndex, (byte)(scaledData.  
15            getByte(scaledIndex) ^ hashedData.getByte(i)));  
16    }  
17  
18    return scaledData;  
19}
```

Listing 2.1: Adressverkürzung im CircleNetworkType (Java)

address

Das Argument **address** referenziert ein **Address**-Objekt, welches den als Grundlage für eine Adresse verwendeten öffentlichen Schlüssel enthält.

Data

Im Quellcode von DCL definierte Klasse zur Speicherung von Binärdaten.

hash

Die Instanzvariable **hash** referenziert ein **Hash**-Objekt, welches den im Network Type angegebenen Hashalgorithmus implementiert.

byteLength

Die Instanzvariable **byteLength** enthält die Ziellänge der Adressen, in Bytes.

Mit **address.toData()** wird der öffentliche Schlüssel der übergebenen Adresse zuerst in Binärdaten umgewandelt, auf welche dann ein Hashalgorithmus angewandt wird. Entspricht die Ziellänge von Adressen im Network Type der Digest Length des verwendeten Hashalgorithmus, so wird der Hashwert **hashedData** direkt zurückgegeben.

Andernfalls wird der Hashwert aus **hashedData** auf Binärdaten der Länge **byteLength**, gespeichert in **scaledData**, abgebildet und diese zurückgegeben.

2.5. Routing

2.5.1. Notwendigkeit

Da es bereits ab einer geringen Anzahl an Teilnehmern im Peer-to-Peer-Netzwerk nicht mehr praktikabel ist, ein vollständig vermaschtes Netz zu führen, weil die Anzahl an dafür notwendigen Verbindungen die Kapazitäten der Teilnehmer überschreitet, ist es nötig, einen Routingmechanismus für das Netzwerk zu implementieren. Damit können Nachrichten im Peer-to-Peer-Netzwerk von jedem Teilnehmer zu jedem beliebigen anderen Teilnehmer gesendet werden, ohne dass die beiden kommunizierenden Teilnehmer direkt miteinander verbunden sein müssen.

Dieses Unterkapitel beschreibt das Routing im Circle Network.

2.5.2. Schwierigkeit

Im Gegensatz zu herkömmlichen Computernetzwerken sind die Verbindungen zwischen den einzelnen Teilnehmern in einem Peer-to-Peer-Netzwerk variabel und ändern sich ständig. Die Wege für Pakete statisch vorzugeben funktioniert deshalb nicht, stattdessen ist es notwendig, dynamisches Routing zu implementieren.

Die meisten im Einsatz befindlichen Verfahren zum dynamischen Routing betrachten die gesamte Topologie und suchen darin Wege für Pakete. Bei zu großen Topologien werden Teile des Netzwerks zusammengefasst und die Teilnehmer innerhalb dieses Teils von außerhalb als ein einziger Teilnehmer betrachtet. Der Pfadfindungsprozess kann dann effizienter gestaltet werden, indem das Routing in zwei Schritten erfolgt: zuerst außerhalb bis zum zusammengefassten Teil und anschließend innerhalb des zusammengefassten Teils des Netzwerks, wo das Routing zum endgültigen Ziel erfolgt.

Diese Technik funktioniert jedoch nur in einem hierarchisch aufgebauten Netzwerk, in dem Teilnehmer mit ähnlichen Adressen logisch nah beieinander sind, wie das beispielsweise in einem IPv4-Subnetz der Fall ist. In einem dezentralen Peer-to-Peer-Netzwerk, dessen Topologie sich ständig ändert und in dem die Adresse eines Teilnehmers nicht mit seiner Position in der Topologie zusammenhängt, können keine Teilnehmer zusammengefasst werden.

Gleichzeitig ist die Topologie des Peer-to-Peer-Netzwerk so groß und ändert sich so oft, dass es praktisch unmöglich ist, bei jedem Teilnehmer des Netzwerks eine Kopie aller bestehenden Verbindungen zu speichern und diese aktuell zu halten, um basierend darauf die Wege zur Weiterleitung von Nachrichten zu finden.

2.5.3. Distanzbasiertes Routing

Im Circle Network des DCL werden nicht die Wege anhand der Topologie gesucht, sondern die Topologie anhand der gewünschten Wege aufgebaut. Das bedeutet konkret, dass andere Teilnehmer, mit denen sich ein Teilnehmer verbindet, nicht zufällig gewählt werden, sondern ein Teilnehmer sich tendentiell mit mehr Teilnehmern verbindet, die eine ähnliche Adresse haben, als mit solchen, deren Adresse sich von der eigenen stark unterscheidet.

Ähnlich sind zwei Adressen dann, wenn ihre numerische Differenz gering ist. Dabei gilt es, auch den Überlauf beim Überschreiten des mit der Adresslänge maximal darstellbaren Wertes zu beachten. Nimmt man zur vereinfachten Darstellung eine Adresslänge von 16 Bits an, dann weisen die hexadezimal angegebenen Adressen E539 und 1092 eine Distanz

2. Decentralized Communication Layer

von 2B59 auf, obwohl $E539 - 1092 = D4A7$, weil die Distanz über den Überlauf kürzer ist:
 $10000 - E539 + 1092 = 2B59$.

Des Weiteren werden Teilnehmer, die exakt dieselbe Adresse aufweisen, stets direkt miteinander verbunden, um sicherzustellen, dass Nachrichten, deren Zieladresse auf mehrere Teilnehmer verweist, an alle zugestellt werden. Dass zwei Teilnehmer dieselbe Adresse bekommen ist jedoch erstens extrem unwahrscheinlich und bedeutet zweitens nicht unbedingt, dass ihre Adressen auf den selben öffentlichen Schlüsseln basieren. Siehe dazu 2.4.4 Eindeutigkeit von Adressen, Seite 12.

Dadurch, dass die Teilnehmer des Netzwerks jeweils mehr Verbindungen zu Teilnehmern mit Adressen ähnlich zur eigenen haben, als Verbindungen zu Teilnehmern mit Adressen, die eine hohe Differenz zur eigenen aufweisen, können Nachrichten einfach an den Teilnehmer weitergeleitet werden, dessen Adresse die niedrigste Differenz zur Zieladresse der Nachricht aufweist. Auf diese Weise gelangt die Nachricht, logisch betrachtet, immer näher an ihr Ziel und wird schlussendlich an den Adressat zugestellt.

2.5.4. Umsetzung im Quellcode von DCL

Nachfolgend ist jener Teil des Quellcodes von DCL angeführt, welcher im `CircleNetworkType` zum Routinglookup angewandt wird.

```
1 Nexthops lookup(Data scaledDestinationAddress, Address
2   originAddress, Object originIdentifierObject) {
3
4     Nexthops nexthops = localEndpointRoutes.get(
5       scaledDestinationAddress);
6     if(nexthops != null) {
7
8       Nexthops twinNeighbors = remoteServiceRoutes.get(
9         scaledDestinationAddress);
10      if(twinNeighbors != null) {
11
12        boolean append = true;
13        for(ForwardDestination forwardDestination :
14          twinNeighbors) {
15          if(forwardDestination.getIdentifierObject()
16            == originIdentifierObject) {
17            append = false;
18            break;
19          }
20        }
21
22        if(append) {
23          nexthops.append(twinNeighbors);
24        }
25      }
26    }
27
28    return nexthops;
```

```

24
25     }
26
27     nexthops = remoteServiceRoutes.getClosest(
28         scaledDestinationAddress);
29     if(nexthops == null) {
30         return null;
31     }
32
33     for(ForwardDestination forwardDestination : nexthops) {
34         if(forwardDestination.getIdentifierObject() ==
35             originIdentifierObject) {
36             return null;
37         }
38     }
39
40 }
```

Listing 2.2: Routing im CircleNetworkType (Java)

scaledDestinationAddress

Das Argument `scaledDestinationAddress` referenziert ein `Data`-Objekt, welches die Zieladresse enthält, für die ein Routinglookup durchgeführt werden soll.

originAddress

Das Argument `originAddress` referenziert ein `Address`-Objekt, welches den öffentlichen Schlüssel des Teilnehmers enthält, von welchem die weiterzuleitende Nachricht empfangen wurde.

originIdentifierObject

Das Argument `originIdentifierObject` referenziert ein Objekt, das den Teilnehmer identifiziert, von welchem die weiterzuleitende Nachricht empfangen wurde. Dieses wird benötigt, um sicherzustellen, dass die Nachricht nicht zum selben Teilnehmer weitergeleitet wird, von welchem sie schon empfangen wurde.

Nexthops

Im Quellcode von DCL definierte Klasse zur Speicherung von verbundenen Endpunkten in der Form von `ForwardDestination`-Objekten.

ForwardDestination

Im Quellcode von DCL definierte Klasse zur Speicherung eines verbundenen Endpunkts, an den eine Nachricht weitergeleitet werden kann, inklusive dessen `Address`- und `InterserviceChannel`-Objekten.

2. Decentralized Communication Layer

localEndpointRoutes

Die Instanzvariable `localEndpointRoutes` referenziert ein Objekt, das zu Schlüsseln in Form von `Data`-Objekten Werte in Form von `Nexthops`-Objekten speichert. Dieses wird benutzt, um zu Adressen die entsprechenden `Nexthops`-Objekte finden zu können, an die Nachrichten weitergeleitet werden können. In diesem Fall handelt es sich dabei ausschließlich um Adressen, deren Endpunkte sich am lokalen DCL-Service befinden.

remoteServiceRoutes

Die Instanzvariable `remoteServiceRoutes` referenziert ein Objekt, das zu Schlüsseln in Form von `Data`-Objekten Werte in Form von `Nexthops`-Objekten speichert. Dieses wird benutzt, um zu Adressen die entsprechenden `Nexthops`-Objekte finden zu können, an die Nachrichten weitergeleitet werden können. In diesem Fall handelt es sich dabei ausschließlich um Adressen, deren Endpunkte sich auf direkt verbundenen, aber nicht am lokalen DCL-Service befinden.

Mit `localEndpointRoutes.get(scaledDestinationAddress)` wird zu Beginn des Routinglookups nach lokalen Endpunkten mit der genauen Zieladresse dieser Nachricht gesucht. Werden ein oder mehrere Endpunkte gefunden, so wird weiters geprüft, ob es entfernte Endpunkte gibt, die dieselbe Adresse haben. Gibt es solche Endpunkte, dann wird sichergestellt, dass keiner dieser Endpunkte derjenige Endpunkt ist, von dem die Nachricht empfangen wurde und dass die Nachricht somit nicht wieder zu diesem Endpunkt weitergeleitet wird. Anschließend werden die Endpunkte mit der gleichen Adresse wie der lokale Endpunkt zur Liste der Endpunkte, an die die Nachricht weitergeleitet werden soll, hinzugefügt. Das geschieht, indem das von `twinNexthops` referenzierte `Nexthops`-Objekt mit `nexthops.append(twinNexthops)` an das von `nexthops` referenzierte `Nexthops`-Objekt angehängt wird.

Wurden keine lokalen Endpunkte mit der Zieladresse der Nachricht gefunden, dann wird mit `remoteServiceRoutes.getClosest(scaledDestinationAddress)` nach direkt verbundenen Endpunkten gesucht, deren Adresse der Zieladresse am ähnlichsten ist. Anschließend wird, wie schon bei den lokalen Endpunkten, sichergestellt, dass die Liste von Endpunkten nicht den Endpunkt enthält, von dem die Nachricht empfangen wurde. Falls das der Fall ist, wird `null` zurückgegeben und die Nachricht in Folge verworfen.

Andernfalls wird nur der letzte Endpunkt in der von `nexthops` referenzierten Liste der Endpunkte zurückgegeben: Enthält die Liste mehrere Einträge, so haben die darin enthaltenen Endpunkte dieselbe Adresse. Die Nachricht soll in diesem Fall aber nur an einen dieser Endpunkte weitergeleitet werden, da sie von dort an die restlichen Endpunkte mit der gleichen Adresse weitergeleitet wird.

2.6. NAT-Traversal

2.6.1. Einleitung

Unter Network Address Translation (NAT) versteht man das Übersetzen von Netzwerkadressen. NAT kommt bei praktisch allen Routern privater Internetanschlüsse zum Einsatz und ermöglicht es dort, mehrere Hosts über eine einzige öffentliche IP-Adresse an das Internet anzubinden.

Realisiert wird das durch Übersetzen der privaten Quell-IP-Adressen zur öffentlichen IP-Adresse bei Paketen die das lokale Netzwerk verlassen und Rückübersetzen der öffentlichen

Ziel-IP-Adresse zu den privaten IP-Adressen bei Paketen die aus dem Internet in das lokale Netzwerk eintreten.

Bei ankommenden Paketen wird dabei anhand des Ziel-Ports entschieden, an welche lokale IP-Adresse das Paket weitergeleitet wird. Um es mehreren lokalen Hosts zu ermöglichen, vom gleichen Port aus Pakete zu senden, werden die Quell-Ports von Paketen die das lokale Netzwerk verlassen gegebenenfalls, zusätzlich zur Quell-IP-Adresse, ebenfalls übersetzt. Beim Eintritt von Antwortpaketen in das lokale Netzwerk werden die Ziel-Ports entsprechend Rückübersetzt. Man spricht dann von der sogenannten Port Address Translation, die eine Unterkategorie der Network Address Translation bildet.

Im NAT-Gerät werden die aktuellen Übersetzungen in einer Datenstruktur gespeichert, die lokale IP-Adressen und Ports auf globale IP-Adressen und Ports abbildet. Dabei werden Datenreihen im folgenden Format gespeichert:

(lokale IP-Adresse, lokaler Port, globale IP-Adresse, globaler Port)

Verlässt beispielsweise ein Paket das lokale Netzwerk, das von der lokalen IP-Adresse 10.0.0.1 und vom Port 28785 aus gesendet wurde, könnte die dafür von einem NAT-Gerät mit der öffentlichen IP-Adresse 93.184.216.34 angelegte Datenreihe wie folgt aussehen:

(10.0.0.1, 28785, 93.184.216.34, 28785)

An dieser Stelle sei angemerkt, dass der Quellport des Pakets in diesem Beispiel nicht übersetzt wird. Ist der Host mit der IP-Adresse 10.0.0.1 der einzige im lokalen Netzwerk, der von Port 28785 aus sendet, muss eine Übersetzung des Ports auch nicht zwingend stattfinden. Nimmt man jedoch an, dass sich ein zweiter Host im lokalen Netzwerk befindet, der beispielsweise die IP-Adresse 10.0.0.2 besitzt und ebenfalls von Port 28785 aus sendet, so würde eine weitere Datenreihe angelegt, die wie folgt aussieht:

(10.0.0.2, 28785, 93.184.216.34, 28786)

Für Pakete des Hosts mit der IP-Adresse 10.0.0.2, die das lokale Netzwerk verlassen, wird also nicht nur die Quell-IP-Adresse übersetzt, sondern auch der Quellport von 28785 auf 28786. Dieser Schritt ist notwendig, da sonst bei Paketen, die aus dem Internet in das lokale Netzwerk eintreten, nicht unterschieden werden kann, ob diese an den Host mit der IP-Adresse 10.0.0.1 oder an den Host mit der IP-Adresse 10.0.0.2 weitergeleitet werden sollen. Die Übersetzung des Quellports ermöglicht es, das lokale Ziel von Antwortpaketen anhand deren Zielport zu bestimmen.

2.6.2. Probleme für Peer-to-Peer-Verbindungen

Verbindungen zwischen zwei über NAT-Geräte angebundenen Hosts werden durch Network Address Translation erschwert, da die Pakete zur Initialisierung der Verbindung aufgrund fehlender Übersetzungsinformationen in den NAT-Geräten nicht an den Zielhost weitergeleitet werden können.

Um trotzdem Peer-to-Peer-Verbindungen zwischen solchen Hosts aufzubauen zu können, müssen Verfahren zum NAT-Traversal angewandt werden. Hierfür existieren unterschiedliche Protokolle zur Einrichtung einer Portweiterleitung auf dem lokalen NAT-Gerät, welche eine Verbindung von Außen ermöglicht.

2. Decentralized Communication Layer

Die derzeitige Implementierung des DCL-Service wendet das sogenannte NAT Hole Punching an, für das kein spezielles Netzwerkprotokoll notwendig ist.

2.6.3. NAT Hole Punching

Für NAT Hole Punching wird für eine neue Verbindung ein Kommunikationskanal zwischen den Verbindungspartnern benötigt, der schon vor Aufbau der Verbindung existieren muss. Diesen Kommunikationskanal kann ein dritter Host bereitstellen, zu dem beide Verbindungspartner bereits verbunden sind, oder, wie im Fall von DCL, ein DCL-Netzwerk, über das beide Kommunikationspartner bereits kommunizieren können. In beiden Fällen muss bereits eine Verbindung zu einem Host außerhalb des lokalen Netzwerks bestehen.

Einer der beiden Hosts, die direkt miteinander verbunden werden sollen, ermittelt seine öffentliche IP-Adresse und seinen öffentlichen Port mithilfe eines Hosts, zu dem bereits eine Verbindung besteht. Diese Information wird über den bestehenden Kommunikationskanal an den Host übermittelt, zu dem eine Verbindung aufgebaut werden soll. Dieser sendet an die eben erhaltene Adresse ein Paket, das das lokale Netzwerk zwar verlässt, am NAT-Gerät des Ziels jedoch verworfen wird. Der Grund dafür liegt darin, dass das Ziel, an das es im lokalen Netzwerk der Empfängerseite weitergeleitet werden soll, vom dortigen NAT-Gerät nicht bestimmt werden kann, da kein passender Eintrag in der Datenstruktur der Übersetzungen existiert. Das NAT-Gerät auf der Senderseite dieses Pakets erstellt jedoch trotzdem einen Übersetzungseintrag in seiner Datenstruktur, um Antwortpakete weiterleiten zu können.

Gleichzeitig ermittelt dieser Host über einen anderen Host, zu dem bereits eine Verbindung besteht, ebenfalls seine öffentlichen Adresse. Diese sendet er zusätzlich zu dem soeben in der Übertragung gescheiterten Paket über den bestehenden Kommunikationskanal an den ersten Verbindungspartner.

Dieser sendet, sobald er die öffentliche Adresse des Gegenübers empfangen hat, ein normales Paket zur Verbindungsinitialisierung an dieses Gegenüber. Dieses Paket verlässt das lokale Netzwerk, das lokale NAT-Gerät speichert einen Übersetzungseintrag in seiner Datenstruktur und das Paket erreicht das entfernte NAT-Gerät. Dort wird jener Übersetzungseintrag gefunden, der durch das in der Übertragung bewusst gescheiterte Paket angelegt wurde. Das Paket wird anhand dieses Eintrags an den richtigen Host weitergeleitet und die Hosts können ab diesem Zeitpunkt normal miteinander kommunizieren.

2.6.4. Einschränkungen

Um NAT Hole Punching durchführen zu können, muss jedenfalls bereits mindestens eine Verbindung zu einem Host außerhalb des lokalen Netzwerks existieren. Die erste Verbindung, die ein über ein NAT-Gerät angebundenes Gerät aufbaut, muss deshalb jedenfalls zu einem Host erfolgen, der selbst nicht über ein NAT-Gerät angebunden ist und somit Verbindungen problemlos annehmen kann. Im Fall von DCL sind das DCL-Services, die auf Hosts laufen, die dieses Kriterium erfüllen und zusätzlich über eine statische IP-Adresse verfügen. In der Praxis handelt es sich bei diesen Hosts um Server. Um einen möglichst problemlosen Einstieg in den DCL gewährleisten zu können, ist es nötig, auf eine der Anzahl an Teilnehmern im DCL entsprechend große Menge an DCL-Services zurückgreifen zu können, die auf solchen Hosts laufen.

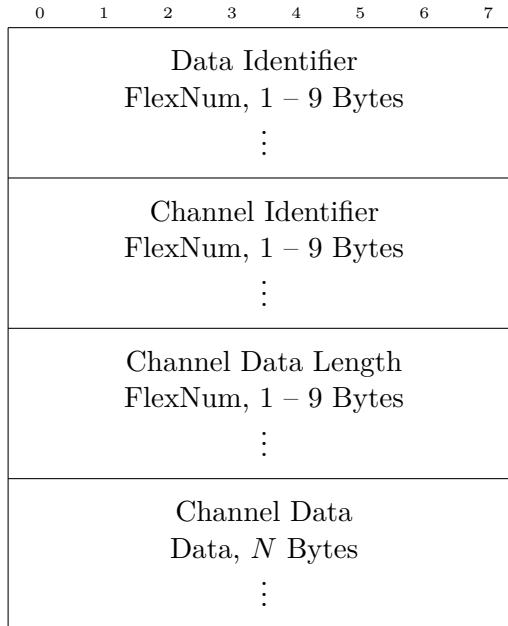


Abbildung 2.1.: Link Packet

2.7. Links

2.7.1. Einleitung

Um NAT-Traversal möglichst problemlos durchführen zu können, erfolgt die Kommunikation zwischen DCL-Services über das User Datagram Protocol (UDP). Da UDP selbst jedoch keine verlässliche Übertragung und keine Verschlüsselung bietet, werden die Datenströme zwischen DCL-Services über Links übertragen. Diese gewährleisten die verlässliche und geordnete Übertragung der Daten, außerdem wird der Übertragungskanal verschlüsselt.

2.7.2. Sicherheit

Links werden mit dem Advanced Encryption Standard (AES) im Galois/Counter Mode (GCM) verschlüsselt. Das garantiert sowohl Vertraulichkeit, Authentizität als auch Integrität der verschlüsselten Daten.

2.7.3. Channels

Über einen Link können mehrere getrennte Datenströme gleichzeitig übertragen werden. Um diese voneinander abzugrenzen, werden sie in unterschiedlichen Channels übertragen.

Auch die zur Verwaltung des Links notwendigen Nachrichten werden in einem Channel, dem Management Channel, übertragen.

Der Link selbst definiert deshalb nur eine einzige Message, die Daten, einen Channel Identifier für die Zuweisung dieser Daten zu einem Channel und einen Data Identifier für die Angabe der Reihenfolge dieser Daten innerhalb des Channels überträgt. Siehe dazu Abbildung 2.1, Link Packet. Für FlexNum-Component, siehe 2.11 FlexNum, Seite 50.

2. Decentralized Communication Layer

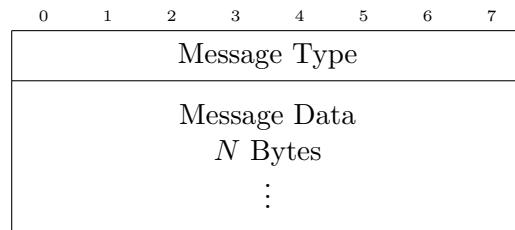


Abbildung 2.2.: BMCP – Genereller Messageaufbau

Die Messages zur Verwaltung des Links definiert das verwendete Management Channel Protocol. Für die Initialisierung des Links werden die Messages des Basic Management Channel Protocol (BMCP) verwendet. BMCP definiert aber auch Messages für den späteren Wechsel des verwendeten Management Channel Protocol.

2.7.4. Basic Management Channel Protocol

2.7.4.1. Einleitung

Das Basic Management Channel Protocol (BMCP) ist das standardmäßig verwendete Management Channel Protocol für Links. Es definiert Messages und Abläufe zum Aufbau der Verbindung, zum Öffnen von Channels, zur erneuten Übertragung von verlorenen Nachrichten und zur Flusskontrolle.

2.7.4.2. Initialisierung

Während der Initialisierung eines Links wird entschieden, wie dieser Link verschlüsselt wird und wie diese Verschlüsselung initialisiert wird. Der Initiator des Links sendet dazu in der Connect Request Message eine Liste an möglichen Crypto Initialization Methods, aus der der Empfänger eine wählt und diese in der Connect Reply Message bestätigt. Danach werden, je nach Crypto Initialization Method unterschiedlich, eine Reihe an Crypto Init Messages gesendet, bis die Verschlüsselung initialisiert ist. Zur Zeit existiert nur eine Crypto Initialization Method: AES/GCM via RSA. Bei dieser Crypto Initialization Method werden, nach Austausch der öffentlichen RSA-Schlüssel, zufällig generierte AES-Schlüssel mit den öffentlichen RSA-Schlüsseln verschlüsselt, übertragen und angewandt.

Für eine spätere Implementierung ist eine zusätzliche Crypto Initialization Method geplant, die die AES-Schlüssel über das Diffie-Hellman-Merkle Schlüsselaustauschverfahren generiert.

2.7.4.3. Messageaufbau

Die Messages des Basic Management Channel Protocol sind generell so aufgebaut, dass ein einzelnes Byte am Beginn der Message deren Typ angibt, anhand dessen die restliche Nachricht interpretiert wird.

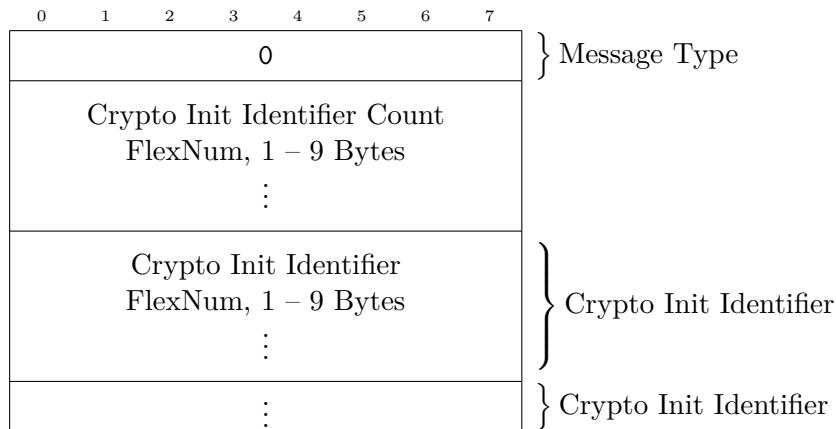


Abbildung 2.3.: BMCP – Connect Request Message

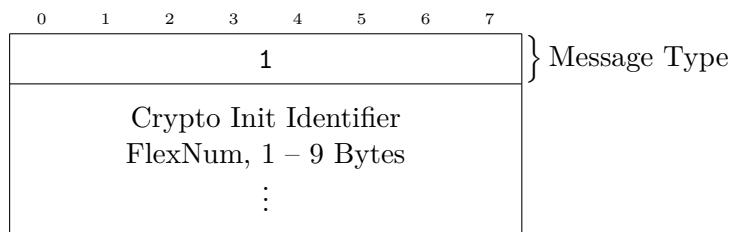


Abbildung 2.4.: BMCP – Connect Reply Message

2.7.4.4. Messages

Connect Request

Die Connect Request Message ist die Message zur Verbindungsanfrage und folglich die erste Message eines Links, die übertragen wird. Die Message enthält eine Liste an Crypto Initialization Method Identifiers jener Crypto Initialization Methods, die der Sender akzeptiert. Der Empfänger antwortet bei Annahme der Verbindung mit einer Connect Reply Message, die einen der angeführten Crypto Initialization Method Identifiers enthält.

Connect Reply

Die Connect Reply Message wird als Antwort auf eine vorhergehende Connect Request Message gesendet und enthält einen der darin angeführten Crypto Initialization Method Identifiers. Dieser bestimmt, wie der Link verschlüsselt wird und wie diese Verschlüsselung initialisiert wird. Zum weiteren Verbindungsauflauf folgt, je nach verwendeter Crypto Initialization Method, eine Reihe an Crypto Init Messages.

Disconnect

Die Disconnect Message dient zur Beendigung des Links. Der Empfänger bestätigt die Auflösung der Verbindung durch Senden einer Kill Message.

2. Decentralized Communication Layer

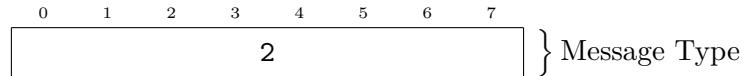


Abbildung 2.5.: BMCP – Disconnect Message

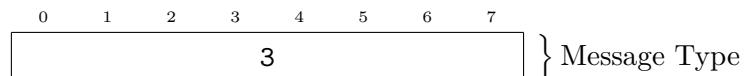


Abbildung 2.6.: BMCP – Kill Message

Kill

Die Kill Message ist die letzte Nachricht eines Links, die übertragen wird und markiert das Ende der Verbindung. Die Message wird als Antwort auf eine Disconnect Message gesendet, kann aber bei unerwartetem Beenden des Links durch einen Verbindungspartner auch alleinstehend gesendet werden.

Crypto Init

Die Crypto Init Message überträgt Daten zur Initialisierung der verwendeten Verschlüsselung des Links und wird zu Beginn der Verbindung, je nach verwandelter Crypto Initialization Method unterschiedlich oft, übertragen.

Ack

Die Ack Message dient zur Bestätigung empfangener Anfragen, wie beispielsweise einer Open Channel Request Message. Die Message enthält den Data Identifier der bestätigten Anfrage.

Change Protocol Request

Die Change Protocol Request Message dient zum Wechsel des verwendeten Management Channel Protocol. Die Message enthält einen Protocol Identifier in Form eines Strings, der das neue Management Channel Protocol angibt. Der Empfänger bestätigt den Wechsel des Protokolls mit einer Ack Message, die den Data Identifier dieser Change Protocol Request Message enthält.

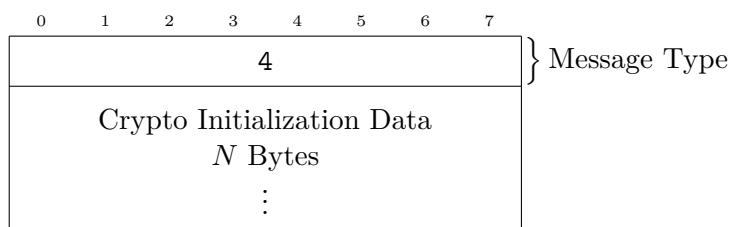


Abbildung 2.7.: BMCP – Crypto Init Message

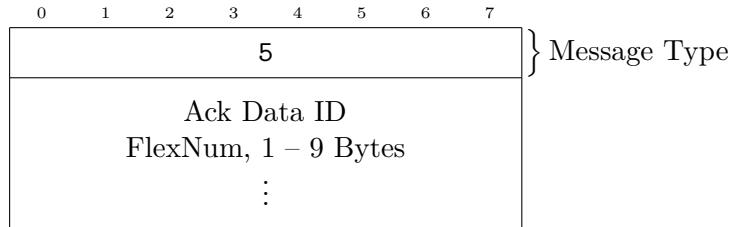


Abbildung 2.8.: BMCP – Ack Message

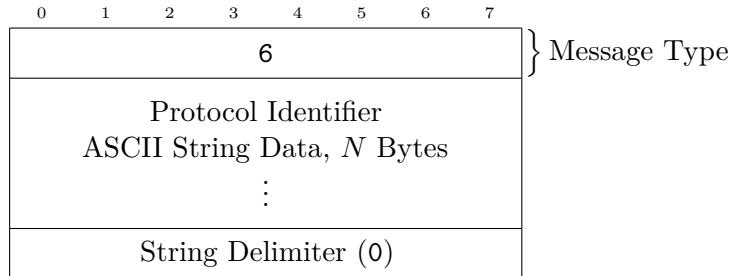


Abbildung 2.9.: BMCP – Change Protocol Request Message

Channel Block Status Request

Die Channel Block Status Request Message dient zur Abfrage der empfangenen Data Identifiers eines oder mehrerer Channels. Die Message enthält eine Liste der Channel Identifiers der Channels, die abgefragt werden sollen.

Der Empfänger antwortet mit einer Channel Block Status Report Message, anhand der der Sender dieser Channel Block Status Request Message feststellen kann, welche Pakete nicht am Ziel angekommen sind und erneut übertragen werden müssen.

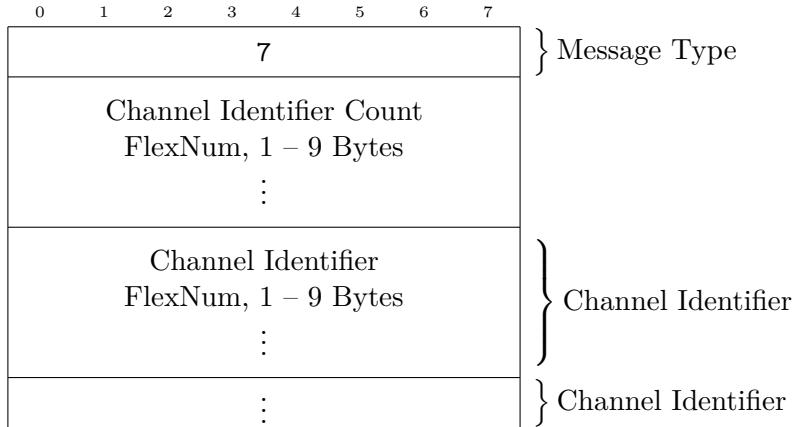


Abbildung 2.10.: BMCP – Channel Block Status Request Message

2. Decentralized Communication Layer

Channel Block Status Report

Die Channel Block Status Report Message wird als Antwort auf eine Channel Block Status Request Message gesendet und enthält eine Liste an Channel Block Status Reports für jeden der in der Anfrage angeführten Channels. Der Empfänger kann anhand dieser Channel Block Status Reports erkennen, welche Pakete nicht übertragen wurden und deshalb erneut gesendet werden müssen.

Ein Channel Block Status Report enthält den Channel Identifier des Channels auf den er sich bezieht, den niedrigsten und den höchsten Data Identifier, die auf diesem Channel empfangen wurden, die Anzahl an unterschiedlichen Data Identifiers, die insgesamt auf diesem Channel empfangen wurden und je eine Liste an nicht empfangenen, einzelnen Data Identifiers sowie an zusammenhängenden Blöcken aus nicht empfangenen Data Identifiers innerhalb des niedrigsten und höchsten Data Identifiers des Channels.

Open Channel Request

Die Open Channel Request Message dient zur Anfrage über das Öffnen eines neuen Channels. Die Message enthält den Channel Identifier des neuen Channels sowie den Protocol Identifier des Protokolls, über das auf dem neuen Channel kommuniziert werden soll.

Nimmt der Empfänger die Anfrage an, wird eine Ack Message gesendet, die den Data Identifier dieser Open Channel Request Message enthält.

Throttle

Die Throttle Message dient zur Limitierung der Sendeübertragungsrate des Empfängers und wird für die Flusskontrolle des Links genutzt. Die Message enthält die maximal zulässige Senderate in Bytes pro Sekunde.

2.8. Interservice-Protokoll

2.8.1. Einleitung

Das Interservice-Protokoll wird zur Kommunikation zwischen zwei direkt miteinander verbundenen DCL-Services verwendet. Es dient zur Übertragung von zwischen Anwendungen gesendeten Nachrichten, zur Bekanntgabe von Endpunkten samt Adressen und beigetretenen Netzwerken, sowie zum Einstieg in Netzwerke des DCL.

Die Messages des Interservice-Protokolls werden über den Interservice Channel, einen Channel des Links zwischen zwei DCL-Services, übertragen. Der Protocol Identifier des Interservice-Protokolls ist `org.dclayer.interservice`.

Dieses Unterkapitel beschreibt das Interservice-Protokoll in Version 0.

2.8.2. Address Slots

Da DCL-Services jeweils mehrere Adressen hosten können, ist es notwendig, diese mit dem Interservice-Protokoll unterscheiden zu können. Jede Adresse wird hierbei einem eigenen Address Slot zugeordnet, um sich in den Messages des Protokolls einfach auf einzelne Adressen beziehen zu können, ohne jedes mal die ganze Adresse senden zu müssen.

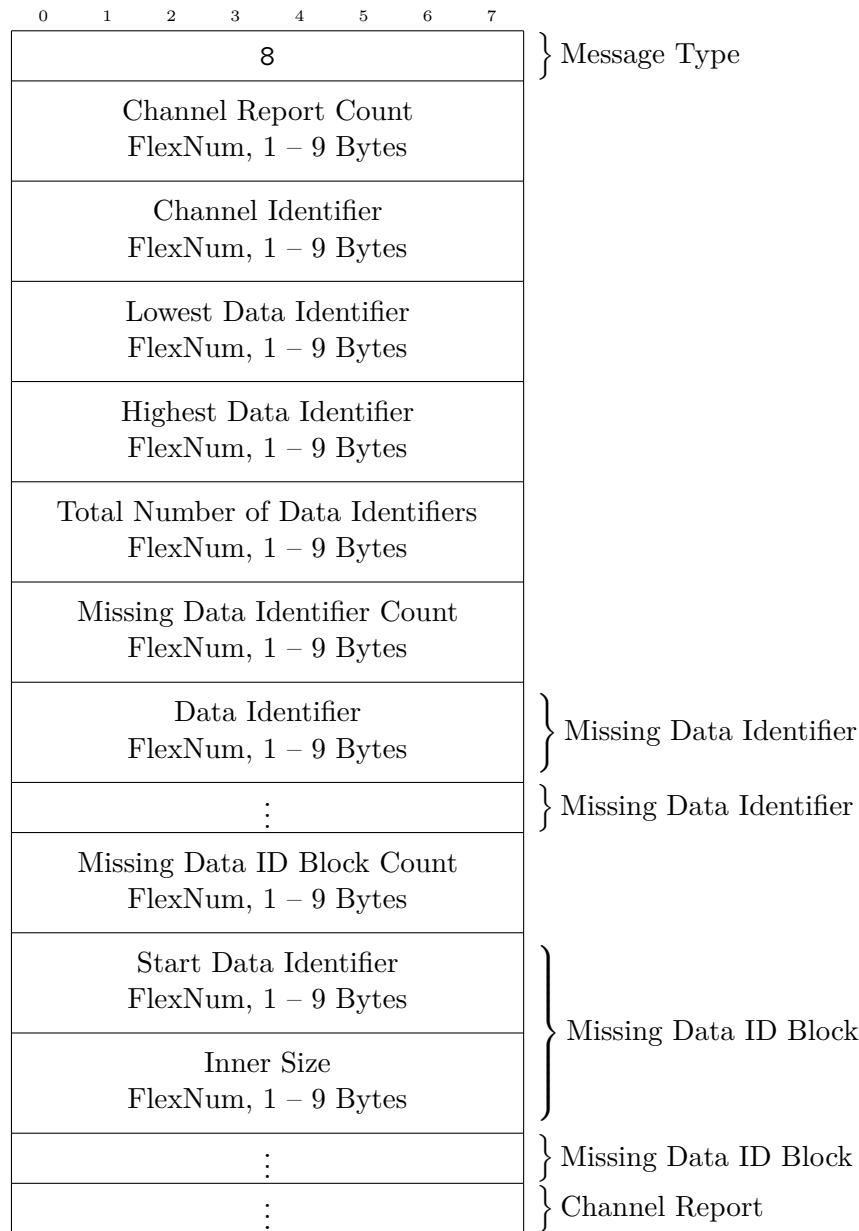


Abbildung 2.11.: BMCP – Channel Block Status Report Message

2. Decentralized Communication Layer

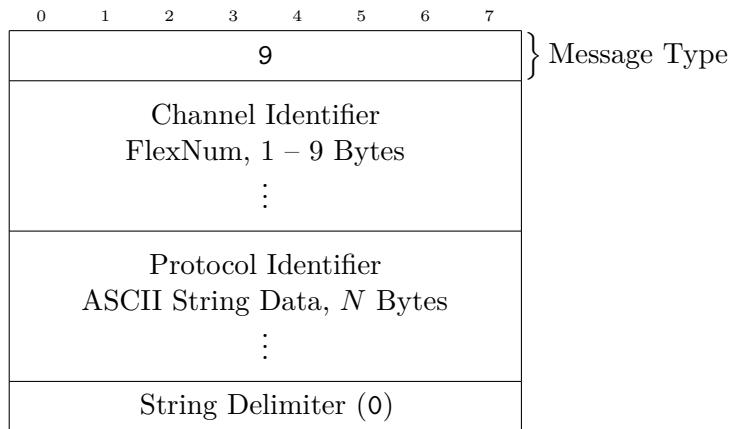


Abbildung 2.12.: BMCP – Open Channel Request Message

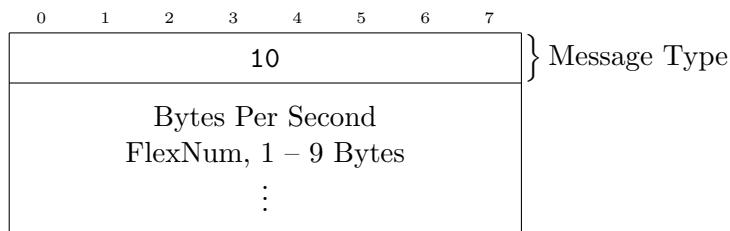


Abbildung 2.13.: BMCP – Throttle Message

2.8.3. Network Slots

Eine Adresse kann gleichzeitig in mehreren verschiedenen Netzwerken vertreten sein. Um diese Netzwerke in den Messages des Interservice-Protokoll einfach referenzieren zu können, ohne jedes mal den ganzen Network Type Identifier übertragen und parsen zu müssen, wird jedes Netzwerk, dem ein DCL-Service mit einer oder mehreren Adressen beitritt einem Network Slot zugeordnet.

Werden Nachrichten zur Kommunikation zwischen Anwendungen, sogenannte Network Packets, über den Interservice Channel übertragen, so wird auch das Netzwerk, in dem diese geroutet werden, in Form des entsprechenden Network Slot angeführt, um beim Empfänger die Zuordnung zu ermöglichen.

2.8.4. Connection Base

Die Connection Base gibt an, welche Messages von einem DCL-Service auf einem Address Slot eines Interservice Channel akzeptiert werden. Derzeit sind zwei Werte für die Connection Base implementiert: **stranger** (0) und **trusted** (1). Ein neuer Address Slot wird immer mit der Connection Base **stranger** initialisiert, unter welcher keine mit dem Routing in Verbindung stehenden Messages akzeptiert werden. Erst ab der Connection Base **trusted** werden Messages akzeptiert, die auf das Routing Einfluss nehmen. Konkret handelt es sich dabei um die Network Join Notice Message, die Network Leave Notice Message, die Network Packet Message, die Integration Request Message, die Application

Channel Slot Assign Message und die Application Channel Data Message.

Als Voraussetzung für die Connection Base **trusted** gilt der Beweis der Adresse dieses Address Slots. Ein DCL-Service kann den Wechsel auf die Connection Base **trusted** für einen existenten Address Slot anfordern, indem er eine Trusted Switch Message an sein Gegenüber sendet. Der entfernte DCL-Service fordert den lokalen DCL-Service danach durch Senden einer Crypto Challenge Request Message auf, mit seiner Adresse eine Crypto Challenge zu lösen, um ihre Echtheit zu beweisen. Siehe dazu 2.4.3 Überprüfung von Adressen, Seite 11. Der lokale DCL-Service antwortet mit einer Crypto Challenge Reply Message und übermittelt die Daten der gelösten Crypto Challenge an den entfernten DCL-Service. Ist die Lösung der Crypto Challenge korrekt, so wird die Connection Base des Address Slots auf **trusted** gesetzt und der lokale DCL-Service mit einer Connection Base Notice Message über die geänderte Connection Base benachrichtigt.

Erst jetzt akzeptiert der entfernte DCL-Service vom lokalen DCL-Service gesendete Network Join Notice Messages und Network Leave Notice Messages.

Die Aufforderung zum Weiterleiten von Network Packets durch die Network Packet Message wird nur akzeptiert, wenn der sendende DCL-Service dem Netzwerk, das vom Network Slot in der Network Packet Message referenziert wird, zuvor mit mindestens einer Adresse beigetreten ist. Das erfordert wiederum die Übertragung einer Network Join Notice Message, für welche die Connection Base **trusted** notwendig ist.

Die Anforderung von Lower Level Addresses (LLAs) geeigneter Nachbarn kann vom Gegenüber ebenfalls erst dann bearbeitet werden, wenn der sendende DCL-Service zuvor zumindest einem Netzwerk mit zumindest einer Adresse beigetreten ist. Das erfordert ebenfalls wiederum die Übertragung einer Network Join Notice Message, für welche die Connection Base **trusted** notwendig ist.

Damit die Aufforderung zum Weiterleiten von in einem Application Channel übertragenen Daten akzeptiert werden kann, muss der in der Application Channel Data Message angeführte Application Channel Slot existieren. Dafür muss dieser zuvor mit einer Application Channel Slot Assign Message angelegt worden sein, wofür ebenfalls die Connection Base **trusted** notwendig ist.

2.8.5. Initialisierung

Die Kommunikation über das Interservice-Protokoll wird mit der Aushandlung einer Protokollversion begonnen. Dazu werden, beginnend mit dem Initiator, also dem DCL-Service, der die Verbindung angefordert hat, so lange Version Messages abwechselnd zwischen den DCL-Services gesendet, bis die letzten zwei übertragenen Version Messages die selben Versionsnummern enthalten.

2.8.6. Messageaufbau

Eine Message im Interservice-Protokoll ist generell so aufgebaut, dass ein FlexNum-Component am Anfang der Message den Typ angibt, anhand dessen die restliche Nachricht interpretiert wird.

FlexNum-Components werden unter 2.11 FlexNum auf Seite 50 beschrieben.

2. Decentralized Communication Layer

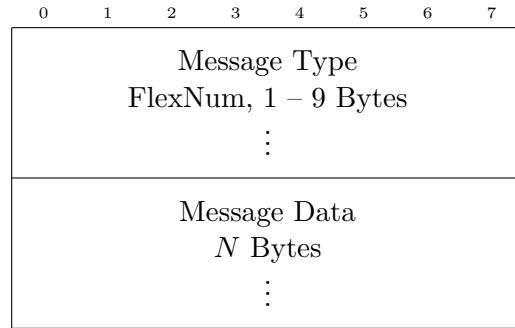


Abbildung 2.14.: Interservice-Protokoll – Genereller Messageaufbau

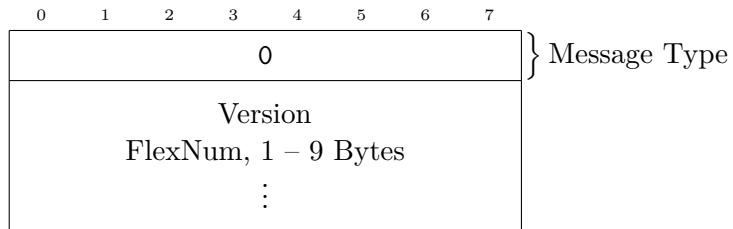


Abbildung 2.15.: Interservice-Protokoll – Version Message

2.8.7. Messages

2.8.7.1. Version

Die Version Message enthält eine Versionsnummer in Form eines FlexNum-Components und wird benutzt, um am Beginn der Verbindung die verwendete Version des Interservice-Protokoll auszuhandeln.

2.8.7.2. LLA Request

Die LLA Request Message wird benutzt, um vom Gegenüber eine Liste an LLAs von anderen DCL-Services anzufordern. Die Message enthält ein Feld für die Maximalanzahl an LLAs, die in der LLA Reply Message zurückgesendet werden sollen.

Diese Message stellt keine Anforderungen an die aktuelle Connection Base.

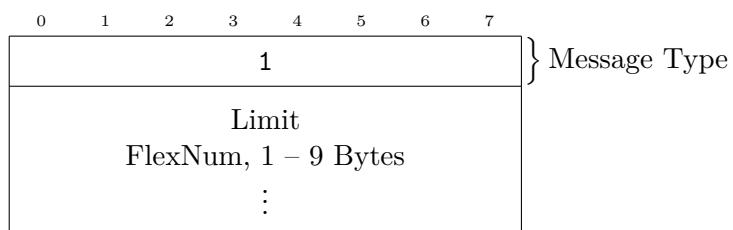


Abbildung 2.16.: Interservice-Protokoll – LLA Request Message

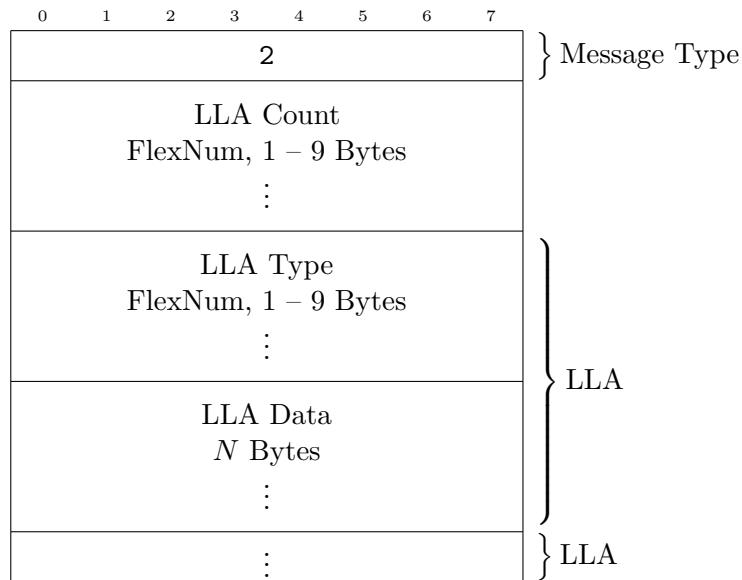


Abbildung 2.17.: Interservice-Protokoll – LLA Reply Message

2.8.7.3. LLA Reply

Die LLA Reply Message wird als Antwort auf eine zuvor empfangene LLA Request Message gesendet und enthält LLAs von anderen DCL-Services.

2.8.7.4. Trusted Switch

Die Trusted Switch Message wird benutzt, um vom Gegenüber eine Crypto Challenge für den als Adresse verwendeten, angegebenen öffentlichen Schlüssel anzufordern. Dieser Schritt wird benötigt, um dem Gegenüber beweisen zu können, dass die angegebene Adresse echt ist. Erst wenn die Crypto Challenge erfolgreich durchgeführt wurde, kann der Sender der Trusted Switch Message damit am Routing jener Netzwerke teilnehmen, denen er mit der angegebenen Adresse beigetreten ist.

Mit dieser Nachricht wird gleichzeitig die in Form eines öffentlichen Schlüssel angegebene Adresse mit dem angegebenen Address Slot assoziiert.

Diese Message stellt keine Anforderungen an die aktuelle Connection Base.

2.8.7.5. Crypto Challenge Request

Die Crypto Challenge Request Message wird als Antwort auf eine Trusted Switch Message gesendet und fordert das Gegenüber auf, die enthaltenen Binärdaten im Rahmen einer Crypto Challenge mit dem zur angegebenen Adresse zugehörigen privaten Schlüssel zu signieren.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn sie zuvor durch Senden einer Trusted Switch Message angefordert wurde.

2. Decentralized Communication Layer

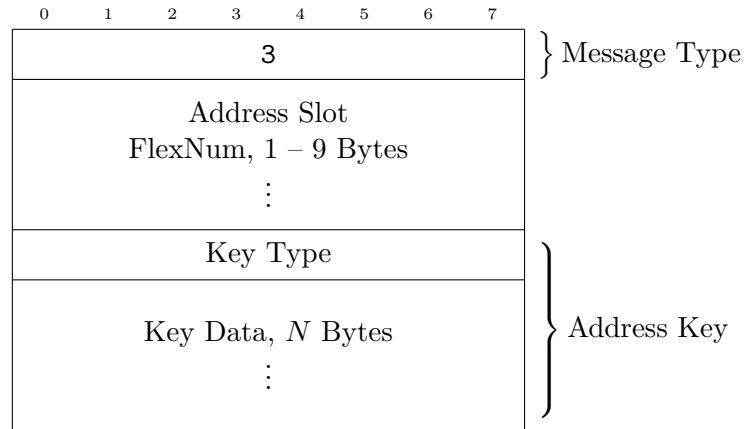


Abbildung 2.18.: Interservice-Protokoll – Trusted Switch Message

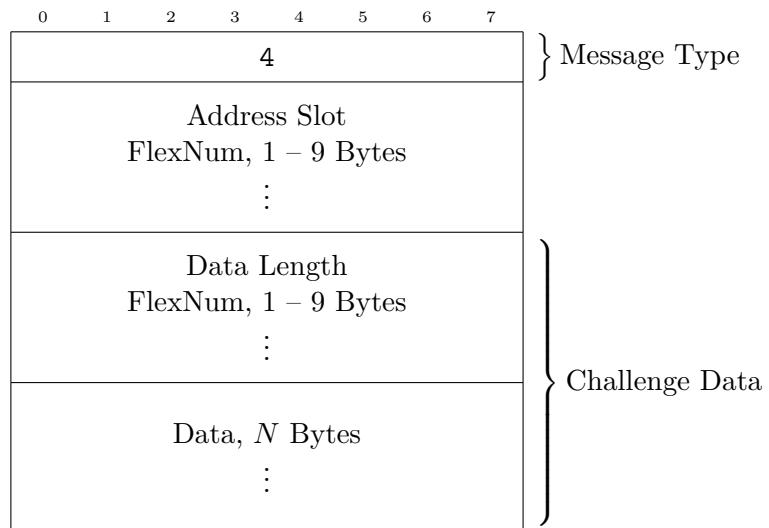


Abbildung 2.19.: Interservice-Protokoll – Crypto Challenge Request Message

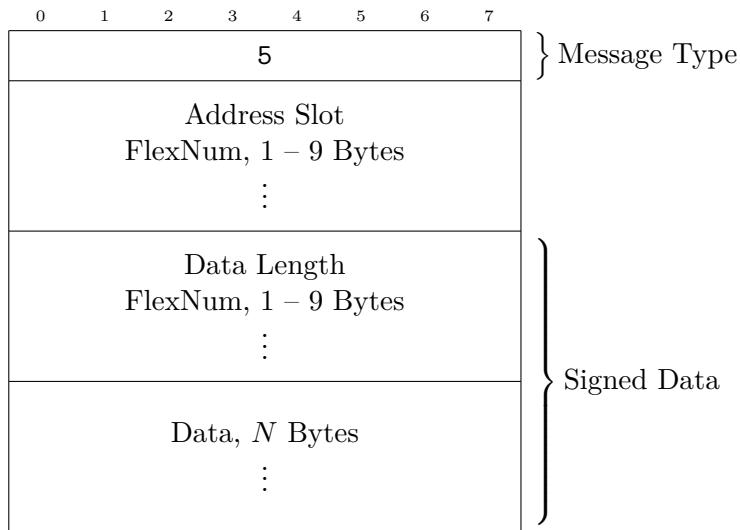


Abbildung 2.20.: Interservice-Protokoll – Crypto Challenge Reply Message

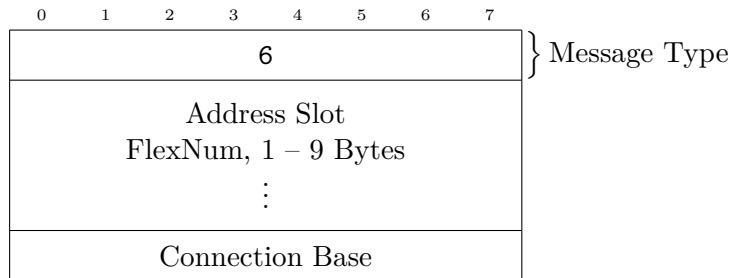


Abbildung 2.21.: Interservice-Protokoll – Connection Base Notice Message

2.8.7.6. Crypto Challenge Reply

Die Crypto Challenge Reply Message wird als Antwort auf eine Crypto Challenge Request Message gesendet und enthält die Daten der gelösten Crypto Challenge.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn sie zuvor durch Senden einer Crypto Challenge Request Message angefordert wurde.

2.8.7.7. Connection Base Notice

Die Connection Base Notice Message wird als Antwort auf eine Crypto Challenge Reply Message gesendet und benachrichtigt den Empfänger über die aktuelle Connection Base der vom angegebenen Address Slot referenzierten Adresse.

Diese Message stellt keine Anforderungen an die aktuelle Connection Base und kann auch gesendet werden, ohne vorher angefordert worden zu sein.

2.8.7.8. Network Join Notice

Die Network Join Notice Message benachrichtigt den Empfänger darüber, dass der sendende DCL-Service mit der vom angegebenen Address Slot referenzierten Adresse dem vom

2. Decentralized Communication Layer

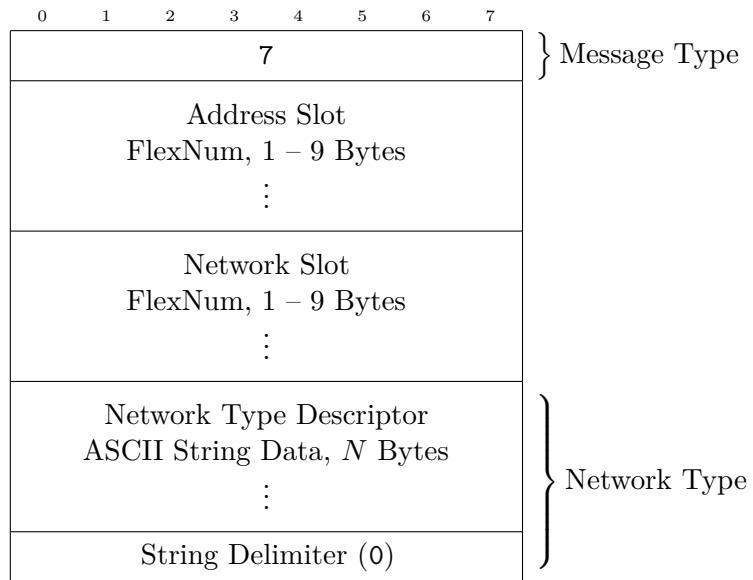


Abbildung 2.22.: Interservice-Protokoll – Network Join Notice Message

angegebenen Network Slot referenzierten Netzwerk beigetreten ist. Sollte der sendende DCL-Service diesem Netzwerk zuvor noch mit keiner Adresse beigetreten sein, so wird dieser Network Slot mit dieser Message angelegt. Das von diesem Network Slot referenzierte Netzwerk wird durch den angegebenen Network Type Identifier spezifiziert. Der Empfänger wird in diesem Fall mit dieser Nachricht außerdem darüber in Kenntnis gesetzt, dass ab jetzt Nachrichten zum Routing in diesem Netzwerk an den sendenden DCL-Service übermittelt werden können. Das Feld des Network Type Identifier bleibt in den folgenden Network Join Notice Messages, die sich auf den angegebenen Network Slot beziehen, leer.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn die Connection Base des angegebenen Address Slot mindestens **trusted** entspricht.

2.8.7.9. Network Leave Notice

Die Network Leave Notice Message benachrichtigt den Empfänger darüber, dass der sendende DCL-Service mit der vom angegebenen Address Slot referenzierten Adresse aus dem vom angegebenen Network Slot referenzierten Netzwerk austritt. Sollte der sendende DCL-Service aktuell mit keiner anderen Adresse diesem Netzwerk beigetreten sein, so wird der Empfänger mit dieser Nachricht außerdem darüber in Kenntnis gesetzt, dass ab jetzt keine Nachrichten mehr zum Routing in diesem Netzwerk an den Sender übermittelt werden können.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn die Connection Base des angegebenen Address Slot mindestens **trusted** entspricht.

2.8.7.10. Integration Request

Die Integration Request Message fordert den Empfänger auf, den sendenden DCL-Service in die gemeinsamen Netzwerke zu integrieren. Konkret bedeutet das, dafür zu sorgen, dass

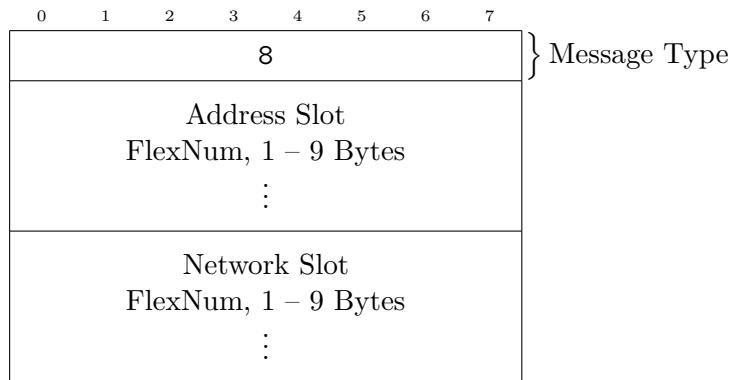


Abbildung 2.23.: Interservice-Protokoll – Network Leave Notice Message

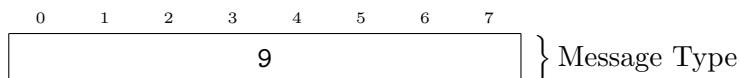


Abbildung 2.24.: Interservice-Protokoll – Integration Request Message

der sendende DCL-Service so mit anderen DCL-Services verbunden wird, dass das Routing der Adressen des sendenden DCL-Service möglichst effizient funktionieren kann.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn der sendende DCL-Service zuvor mit mindestens einer Adresse mindestens einem Netzwerk beigetreten ist.

2.8.7.11. Integration Connect Request

Die Integration Connect Request Message fordert den Empfänger auf, eine Verbindung zur angegebenen LLA aufzubauen und wird als Antwort auf eine frühere Integration Request Message gesendet. Sie wird benötigt, um neue Teilnehmer eines Netzwerks mit ihren entsprechenden Nachbarn verbinden zu können.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn sie zuvor durch Senden einer Integration Request Message angefordert wurde.

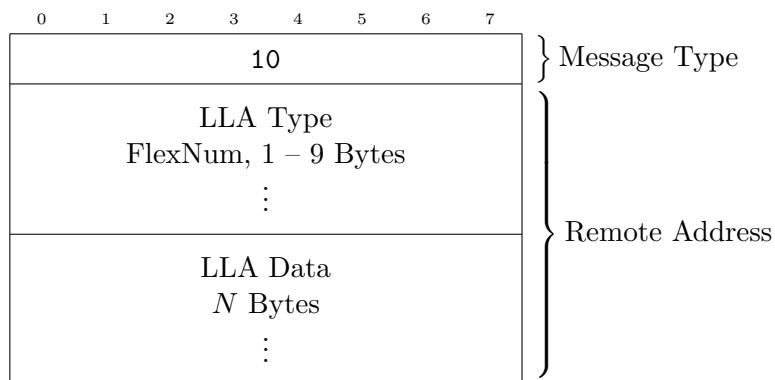


Abbildung 2.25.: Interservice-Protokoll – Integration Connect Request Message

2. Decentralized Communication Layer

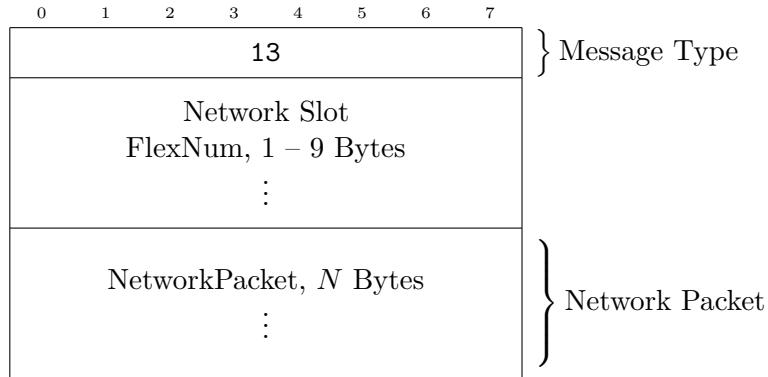


Abbildung 2.26.: Interservice-Protokoll – Network Packet Message

2.8.7.12. Network Packet

Die Network Packet Message enthält ein Network Packet, das vom Empfänger in dem durch den angegebenen Network Slot referenzierten Netzwerk weitergeleitet werden soll.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn der sendende DCL-Service zuvor mit mindestens einer Adresse dem durch den angegebenen Network Slot referenzierten Netzwerk beigetreten ist.

2.8.7.13. Application Channel Slot Assign

Die Application Channel Slot Assign Message benachrichtigt den Empfänger darüber, dass Nachrichten eines Application Channel zwischen den beiden durch die angegebenen Address Slots referenzierten Adressen auf dem angegebenen Application Channel Slot entgegen genommen werden.

Application Channels sind verlässliche Kommunikationkanäle zwischen zwei Anwendungen. Siehe dazu 2.10 Application Channels, Seite 48.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn die Connection Base des angegebenen Address Slot auf dem sendenden DCL-Service mindestens `trusted` entspricht und der angegebene Address Slot auf dem Empfänger existiert.

2.8.7.14. Application Channel Data

Die Application Channel Data Message wird dazu benutzt, um Daten des durch den angegebenen Application Channel Slot referenzierten Application Channel zu übertragen.

Diese Message wird vom Empfänger nur dann akzeptiert, wenn er den angegebenen Application Channel Slot zuvor mit einer Application Channel Slot Assign Message angelegt hat.

2.9. Application-to-Service-Protokoll

2.9.1. Einleitung

Das Application-to-Service-Protokoll wird zur Kommunikation zwischen DCL-Services und über DCL kommunizierenden Anwendungen verwendet. Es dient zur Bekanntgabe von

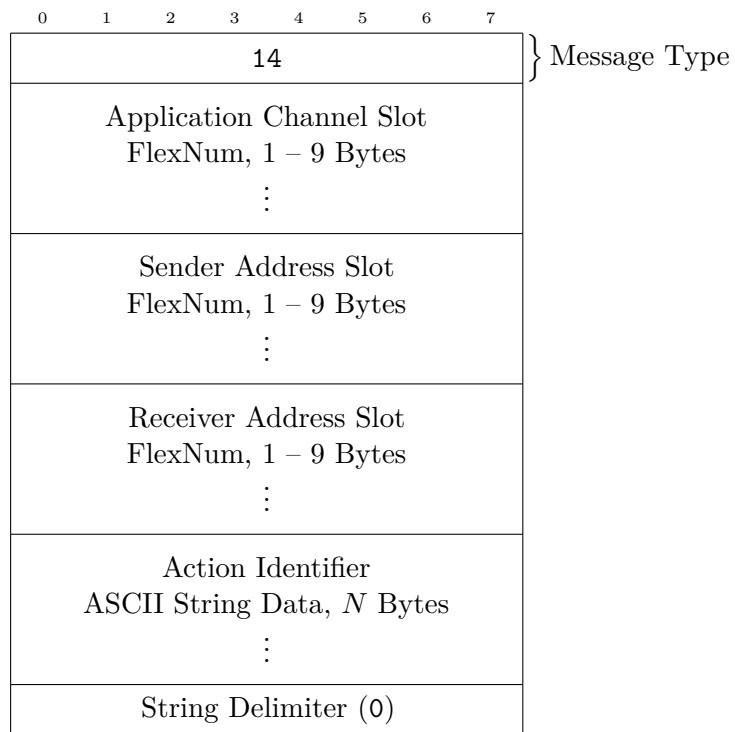


Abbildung 2.27.: Interservice-Protokoll – Application Channel Slot Assign Message

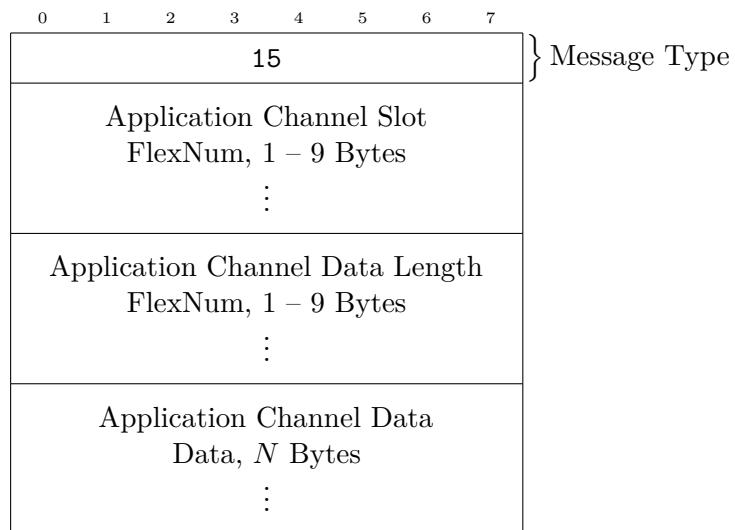


Abbildung 2.28.: Interservice-Protokoll – Application Channel Data Message

2. Decentralized Communication Layer

öffentlichen Schlüsseln, die als Adresse verwendet werden, zum Beitritt zu Netzwerken, zum Senden von verbindungslosen Paketen, zur Verbindung von Application Channels und zur Übertragung von über Application Channels gesendeten Nutzdaten.

Application Channels sind verlässliche Kommunikationkanäle zwischen zwei Anwendungen. Siehe dazu 2.10 Application Channels, Seite 48.

Dieses Unterkapitel beschreibt das Application-to-Service-Protokoll in Revision 0.

2.9.2. Sicherheit

Das Application-to-Service-Protokoll ist zur Kommunikation zwischen einem DCL-Service und einer Anwendung, die auf der selben Maschine wie der DCL-Service läuft, vorgesehen. Die Sicherheitsvorkehrungen des Protokolls sind für den Einsatz auf Verbindungen basierend auf dem Transmission Control Protocol (TCP), die nur über das Loopback-Interface des lokalen Rechners und nicht über ein Netzwerk übertragen werden, angepasst. Über das Application-to-Service-Protokoll werden unter anderem Anweisungen zum Verschlüsseln von Daten gegeben. Bei solch einer Anweisung kommt es zuerst zur Klartextübertragung dieser Daten und kurze Zeit später zur erneuten Übertragung dieser Daten in verschlüsselter Form. Die Möglichkeit, eine Application-to-Service-Verbindung abzuhören, würde ein sehr großes Sicherheitsrisiko darstellen, da das Application-to-Service-Protokoll keinerlei Verschlüsselung der Application-to-Service-Verbindung vorsieht. Die Application-to-Service-Verbindung sollte deshalb immer nur auf dem Localhost erfolgen und niemals über ein Netzwerk.

2.9.3. Network Endpoint Slots

Nach dem derzeitigem Entwicklungsstand des Application-to-Service-Protokoll kann eine Anwendung auf einer Application-to-Service-Verbindung zwar nur eine Adresse bekanntgeben, mit dieser Adresse kann jedoch mehreren Netzwerke beigetreten werden. Eine Kombination aus der bekanntgegebenen Adresse und einem Netzwerk, dem die Anwendung mit dieser Adresse beigetreten ist, ergibt einen Network Endpoint. Network Endpoints innerhalb von Application-to-Service-Verbindungen entsprechen den Endpunkten innerhalb von Interservice Channels. Um sich in Messages einfach auf diese Network Endpoints beziehen zu können, werden im Application-to-Service-Protokoll Network Endpoint Slots verwendet. Dabei handelt es sich im Wesentlichen um Zahlen, die jeweils auf genau einen Network Endpoint, also eine Kombination aus Adresse und Netzwerk, verweisen.

2.9.4. Remote Keys

Um zu verhindern, dass der private Schlüssel des Schlüsselpaars, dessen öffentlicher Schlüssel als Adresse verwendet wird, zum DCL-Service übertragen werden muss und trotzdem die zum Adressbeweis nötigen kryptographischen Operationen ausführen zu können, kommen in der Implementierung des DCL sogenannte Remote Keys zur Anwendung.

Dabei führt der DCL-Service die kryptographischen Operationen, die beispielsweise für Crypto Challenges zum Beweis der Echtheit der Adresse notwendig sind, nicht selbst aus, sondern leitet diese an die Anwendung weiter.

Konkret sendet der DCL-Service dazu Key Encrypt Messages zum Verschlüsseln von unverschlüsselten Daten und Key Decrypt Messages zum Entschlüsseln von verschlüsselten Daten an die Anwendung. Diese Antwortet nach Ausführung der Operation mit jeweils

2.9. APPLICATION-TO-SERVICE-PROTOKOLL

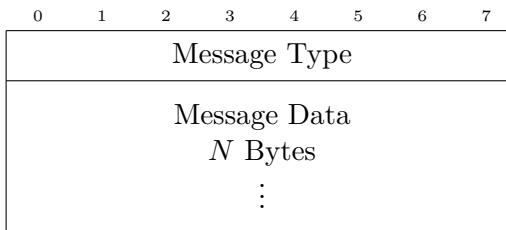


Abbildung 2.29.: Application-to-Service-Protokoll – Genereller Messageaufbau

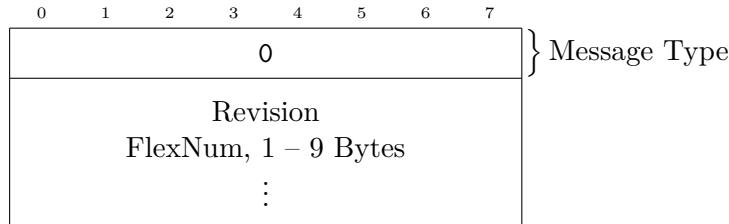


Abbildung 2.30.: Application-to-Service-Protokoll – Revision Message

einer Key Crypto Response Message, die die verschlüsselten bzw. die entschlüsselten Daten enthält.

In der Implementierung des DCL-Service werden Remote Keys dabei als normale Key-Objekte dargestellt, die bei Aufruf der `encrypt()`- und `decrypt()`-Methoden die Anfragen samt Daten an die Anwendung weiterleiten und die Kommunikation in einem separaten Thread abwickeln. Der Methodenaufruf blockt dabei so lange, bis die Anfrage von der Anwendung bearbeitet wurde.

2.9.5. Initialisierung

Die Kommunikation über das Application-to-Service-Protokoll wird mit der Aushandlung der zu verwendenden Revision des Protokolls begonnen. Dazu werden, beginnend mit der Anwendung, so lange Revision Messages abwechselnd zwischen DCL-Service und Anwendung gesendet, bis die letzten zwei übertragenen Revision Messages die selben Revisionsnummern enthalten.

2.9.6. Messageaufbau

Eine Message im Application-to-Service-Protokoll ist generell so aufgebaut, dass ein einzelnes Byte am Anfang der Message den Typ angibt, anhand dessen die restliche Nachricht interpretiert wird.

2.9.7. Messages

2.9.7.1. Revision

Die Revision Message enthält eine Versionsnummer in Form eines FlexNum-Components und wird benutzt, um am Beginn der Verbindung die verwendete Revision des Application-to-Service-Protokoll auszuhandeln.

2. Decentralized Communication Layer

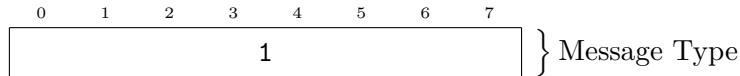


Abbildung 2.31.: Application-to-Service-Protokoll – Generate Key Message

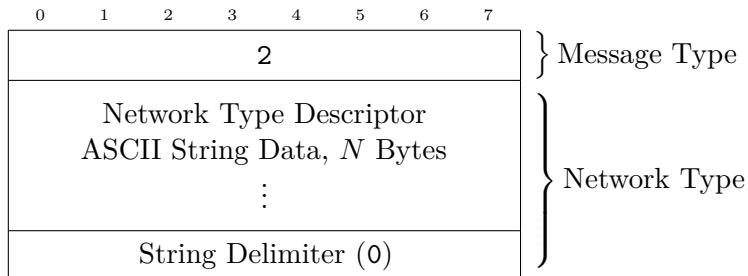


Abbildung 2.32.: Application-to-Service-Protokoll – Join Network Message

2.9.7.2. Generate Key

Die Generate Key Message kann von der Anwendung benutzt werden, wenn keine fixe Adresse notwendig ist und der DCL-Service die Generierung einer temporären Adresse übernehmen soll. Die Übertragung von Messages für kryptographische Funktionen wie der Key Encrypt Message, der Key Decrypt Message oder der Key Crypto Response Message fällt dadurch weg. Die Generate Key Message ersetzt außerdem die Address Public Key Message.

2.9.7.3. Join Network

Die Join Network Message wird von der Anwendung benutzt, um mit der Adresse einem bestimmten Netzwerk beizutreten. Nach erfolgreichem Beitritt antwortet der DCL-Service mit einer Slot Assign Message, die den Network Endpoint Slot für die Adresse und dieses Netzwerk angibt.

2.9.7.4. Slot Assign

Die Slot Assign Message wird vom DCL-Service als Antwort auf eine Join Network Message bzw. Join Default Networks Message gesendet und beinhaltet den Network Endpoint Slot des neu angelegten Network Endpoint, dessen Network Type und die Daten, die innerhalb dieses Netzwerks als skalierte Adresse zum Routing verwendet werden. Letztere würden im Fall des Circle Network also den Daten entsprechen, die durch das unter 2.4.5.2 Adressverkürzung auf Seite 12 angeführte Codebeispiel generiert werden.

2.9.7.5. Data

Die Data Message wird sowohl von DCL-Service als auch Anwendung benutzt, um verbindungslose Pakete zu übertragen. Die Message enthält den Network Endpoint Slot, von dem aus die Nachricht gesendet wird bzw. an den das Paket adressiert ist, je nach Senderichtung der Message die skalierte Ziel- oder Quelladresse des verbindungslosen Pakets und die Nutzdaten des Pakets. Die Adresse entspricht beim Senden der Data Message

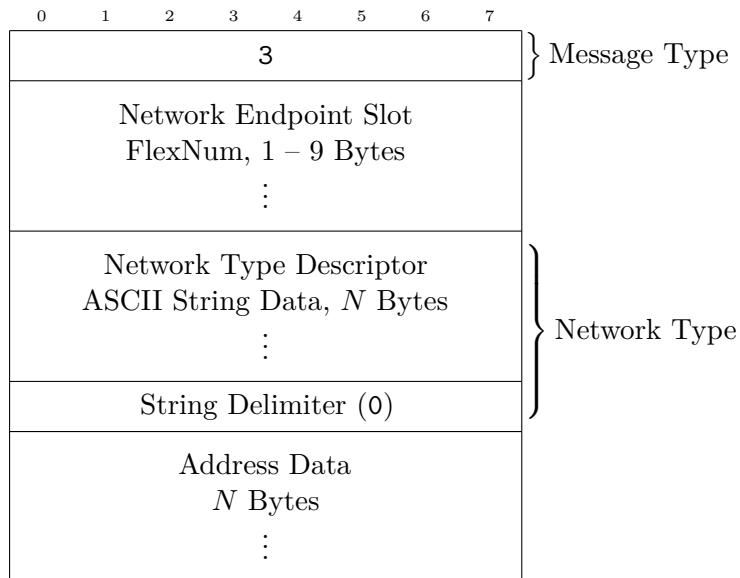


Abbildung 2.33.: Application-to-Service-Protokoll – Slot Assign Message

von der Anwendung zum DCL-Service der Zieladresse, beim Senden vom DCL-Service zur Anwendung der Quelladresse des Pakets.

2.9.7.6. Address Public Key

Die Address Public Key Message wird von der Anwendung benutzt, um den öffentlichen Schlüssel bekanntzugeben, der als Adresse verwendet werden soll. Alternativ kann durch Senden einer Generate Key Message auch der DCL-Service aufgefordert werden, für die Anwendung ein neues, temporäres Schlüsselpaar für die Verwendung als Adresse zu generieren.

2.9.7.7. Join Default Networks

Die Join Default Networks Message kann von der Anwendung gesendet werden, wenn an das Netzwerk keine besonderen Anforderungen gestellt werden und dem gängigsten Netzwerk bzw. den gängigsten Netzwerken beigetreten werden soll. Nach erfolgreichem Beitritt antwortet der DCL-Service mit einer oder mehreren Slot Assign Messages, die die Network Endpoint Slots für die Adresse und die Netzwerke angeben.

2.9.7.8. Key Encrypt

Die Key Encrypt Message wird vom DCL-Service benutzt, um die Anwendung aufzufordern, die enthaltenen Daten mit dem privaten Schlüssel zu verschlüsseln, der aus dem selben Schlüsselpaar stammt wie der als Adresse verwendete und mit der Address Public Key Message bekanntgegebene öffentliche Schlüssel. Die Anwendung übermittelt die verschlüsselten Daten durch Senden einer Key Crypto Response Message.

2. Decentralized Communication Layer

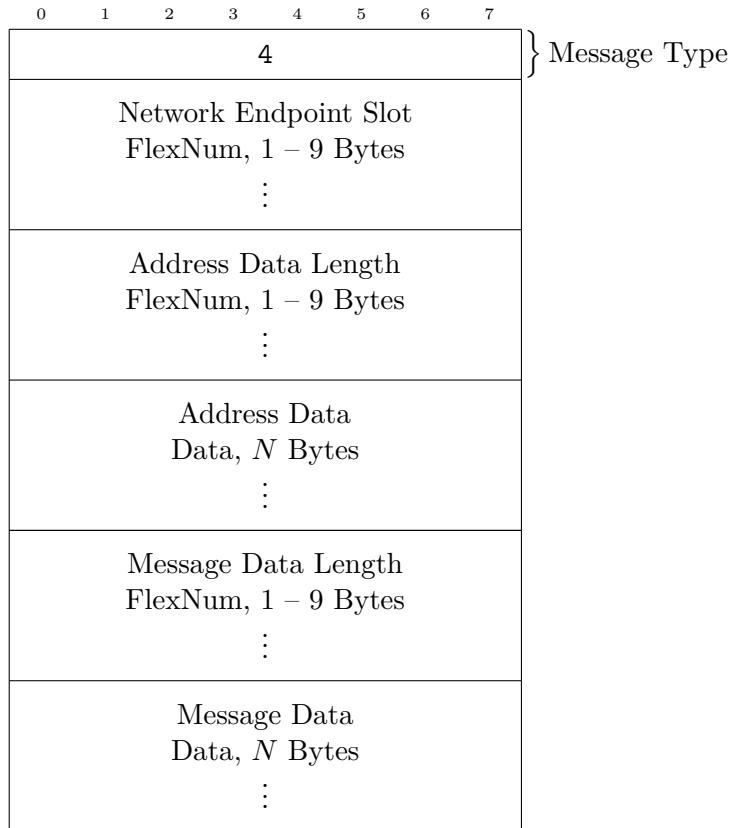


Abbildung 2.34.: Application-to-Service-Protokoll – Data Message

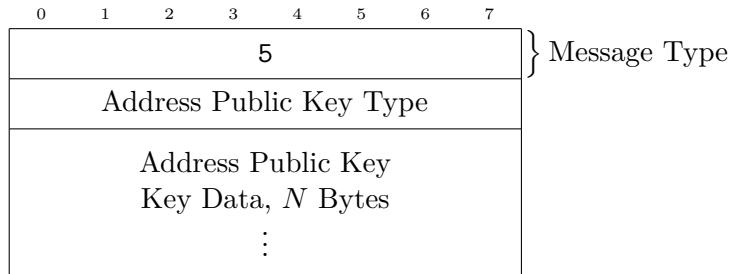


Abbildung 2.35.: Application-to-Service-Protokoll – Address Public Key Message

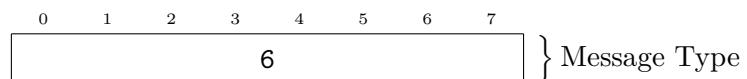


Abbildung 2.36.: Application-to-Service-Protokoll – Join Default Networks Message

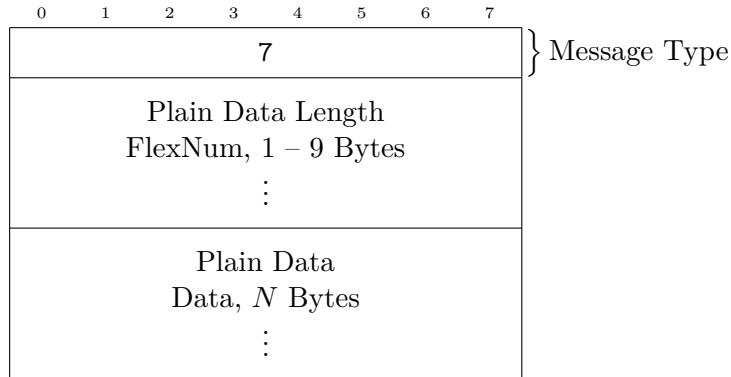


Abbildung 2.37.: Application-to-Service-Protokoll – Key Encrypt Message

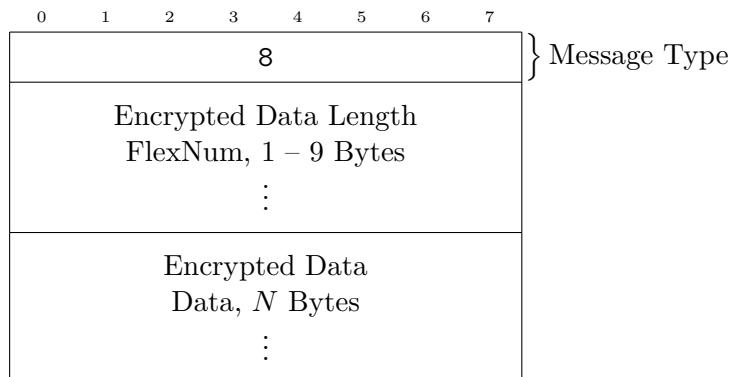


Abbildung 2.38.: Application-to-Service-Protokoll – Key Decrypt Message

2.9.7.9. Key Decrypt

Die Key Encrypt Message wird vom DCL-Service benutzt, um die Anwendung aufzufordern, die enthaltenen Daten mit dem privaten Schlüssel zu entschlüsseln, der aus dem selben Schlüsselpaar stammt wie der als Adresse verwendete und mit der Address Public Key Message bekanntgegebene öffentliche Schlüssel. Die Anwendung übermittelt die entschlüsselten Daten durch Senden einer Key Crypto Response Message.

2.9.7.10. Key Crypto Response

Die Key Crypto Response Message wird von der Anwendung benutzt, um die verschlüsselten Daten einer vorhergehenden Key Encrypt Message bzw. die entschlüsselten Daten einer vorhergehenden Key Decrypt Message zurück an den DCL-Service zu übermitteln.

2.9.7.11. Application Channel Outgoing Request

Die Application Channel Outgoing Request Message wird von der Anwendung benutzt, um einen Application Channel, also eine verlässliche Verbindung zu einer entfernten Anwendung, anzufordern. Die Message überträgt den Network Endpoint Slot, von dem aus der Application Channel angefragt werden soll, den Application Channel Slot, dem

2. Decentralized Communication Layer

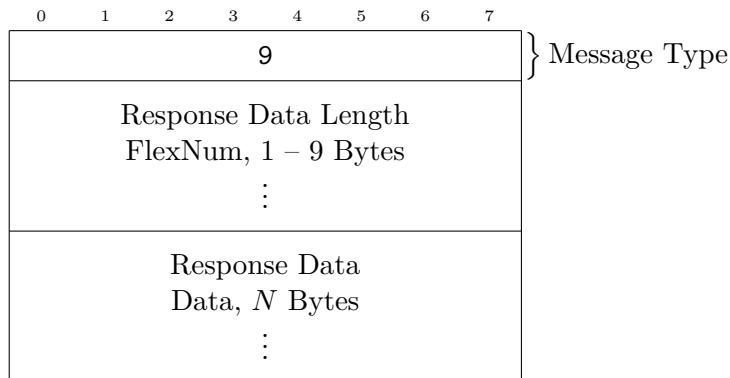


Abbildung 2.39.: Application-to-Service-Protokoll – Key Crypto Response Message

der neue Application Channel zugewiesen werden soll, den Action Identifier des neuen Application Channel und den öffentlichen Schlüssel, der vom Ziel als Adresse benutzt wird. Für Application Channels, siehe 2.10 Application Channels, Seite 48.

2.9.7.12. Application Channel Incoming Request

Die Application Channel Incoming Request Message wird vom DCL-Service benutzt, um die Anwendung über eine eingehende Anfrage über einen neuen Application Channel zu benachrichtigen. Die Message überträgt den Network Endpoint Slot, an den die Anfrage gerichtet ist, den Action Identifier des neuen Application Channel, den öffentlichen Schlüssel, den die Quelle der Anfrage als Adresse verwendet, die LLA des DCL-Service, von dem die Anfrage gesendet wurde und die Ignore Data, die als Präfix für NAT-Traversal-Pakete verwendet werden sollen.

2.9.7.13. Application Channel Accept

Die Application Channel Accept Message wird von der Anwendung benutzt, um eine eingehende Anfrage über einen neuen Application Channel zu bestätigen. Die Message überträgt den Application Channel Slot, dem der neue Application Channel zugewiesen werden soll. Außerdem überträgt die Message eine Kopie der Daten der ihr vorhergehenden Application Channel Incoming Request Message, also den Network Endpoint Slot, an den die ursprüngliche Anfrage gerichtet war, den Action Identifier des neuen Application Channel, den öffentlichen Schlüssel, den die Quelle der ursprünglichen Anfrage als Adresse verwendet, die LLA des DCL-Service, von dem die Anfrage gesendet wurde und die Ignore Data, die als Präfix für NAT-Traversal-Pakete verwendet werden sollen.

2.9.7.14. Application Channel Connected

Die Application Channel Connected Message wird vom DCL-Service benutzt, um die Anwendung darüber in Kenntnis zu setzen, dass der vom angegebenen Application Channel Slot referenzierte Application Channel erfolgreich verbunden wurde und mittels Application Channel Data Message Daten darüber übertragen werden können.

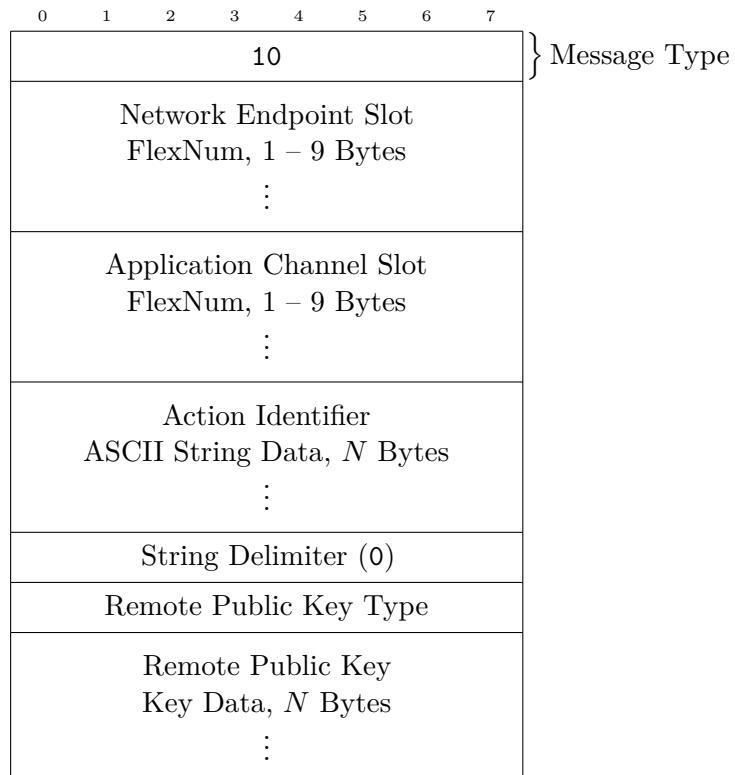


Abbildung 2.40.: Application-to-Service-Protokoll – Application Channel Outgoing Request Message

2. Decentralized Communication Layer

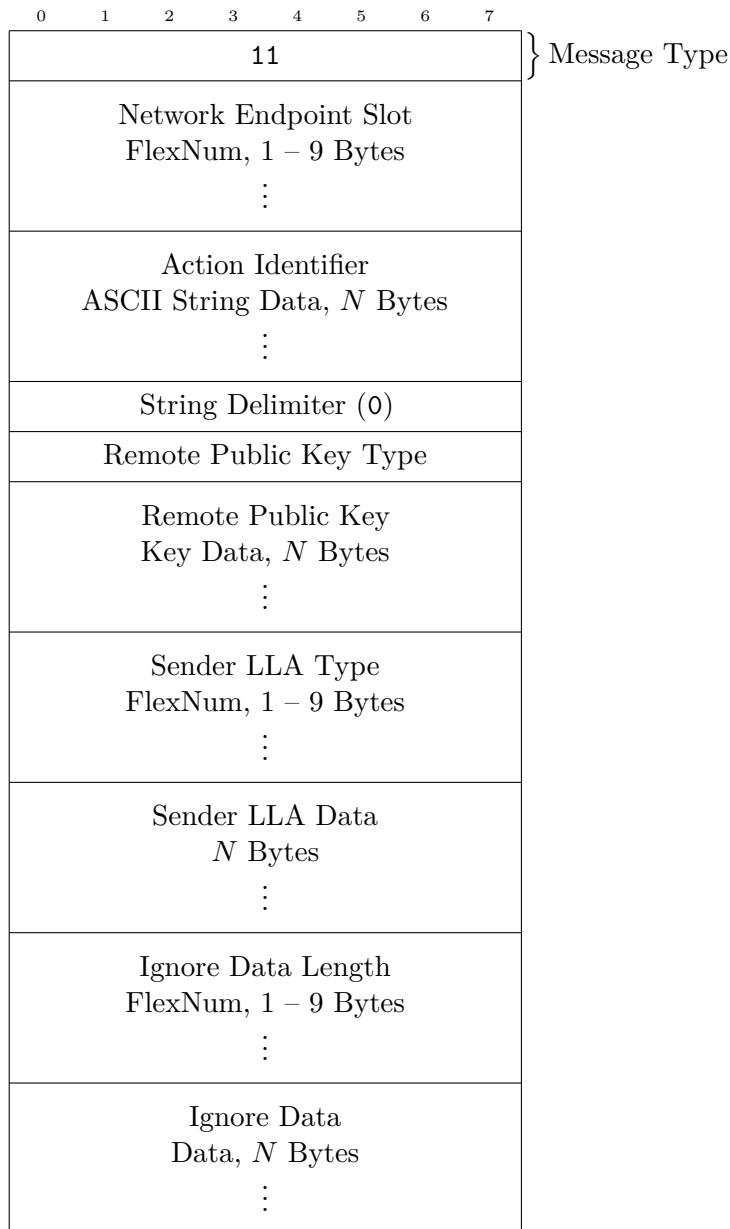


Abbildung 2.41.: Application-to-Service-Protokoll – Application Channel Incoming Request Message

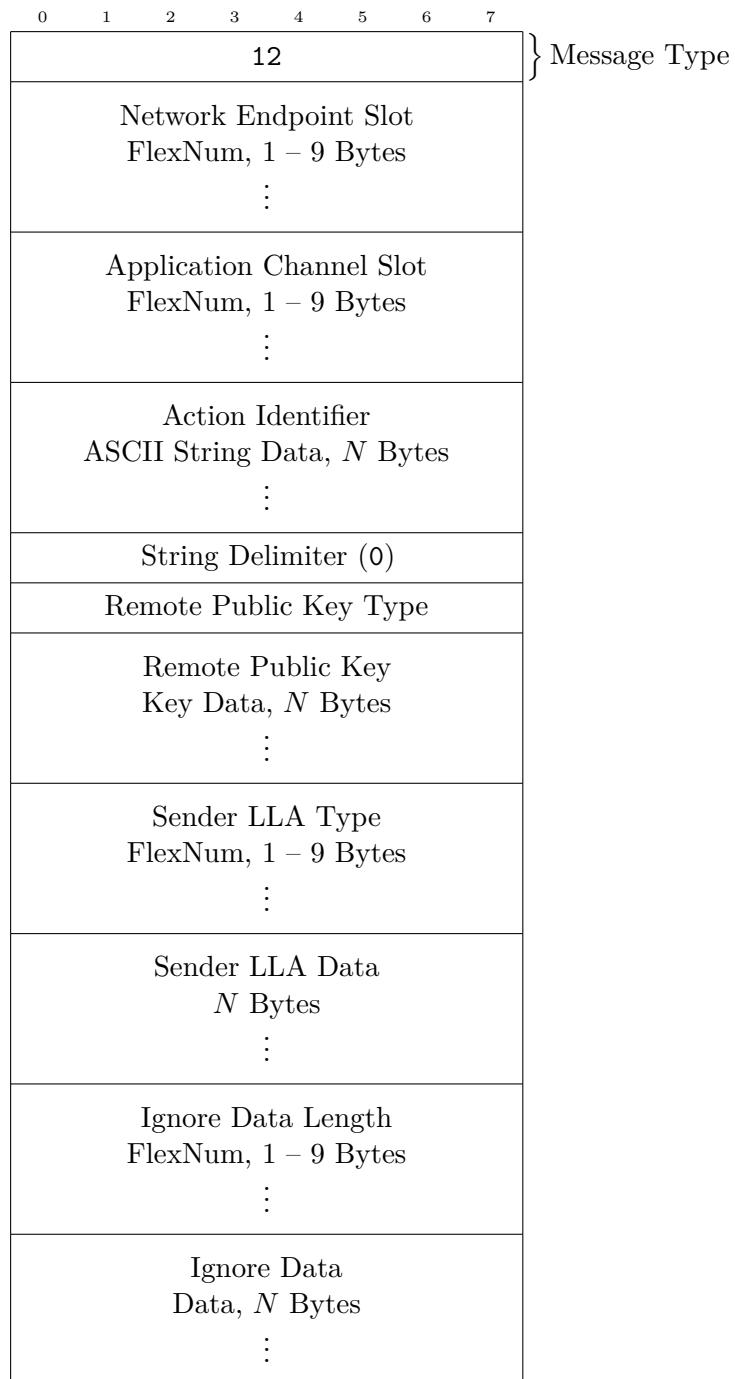


Abbildung 2.42.: Application-to-Service-Protokoll – Application Channel Accept Message

2. Decentralized Communication Layer

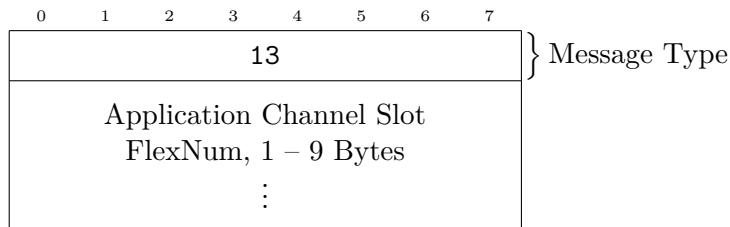


Abbildung 2.43.: Application-to-Service-Protokoll – Application Channel Connected Message

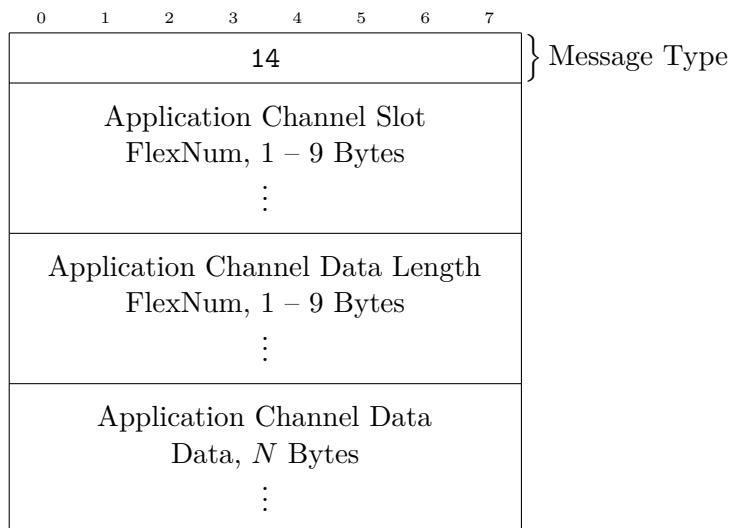


Abbildung 2.44.: Application-to-Service-Protokoll – Application Channel Data Message

2.9.7.15. Application Channel Data

Die Application Channel Data Message wird von sowohl Anwendung als auch DCL-Service benutzt, um Daten aus dem vom angegebenen Application Channel Slot referenzierten Application Channel zu übertragen.

2.9.7.16. Key Encryption Block Size Request

Die Key Encryption Block Size Request Message wird vom DCL-Service benutzt, um von der Anwendung die maximal zulässige Datenmenge abzufragen, die mit dem als Adresse angegebenen öffentlichen Schlüssel in einem Zug verschlüsselt werden kann. Die Anwendung antwortet unmittelbar darauf mit einer Key Number Response Message, die diese Datenmenge in Bytes enthält.

2.9.7.17. Key Number Response

Die Key Number Response Message wird von der Anwendung benutzt, um auf Anfragen bezüglich Zahlenwerte des als Adresse verwendeten öffentlichen Schlüssel zu antworten. Der Zahlenwert wird in Form eines FlexNum-Component übertragen. Derzeit ist die einzige

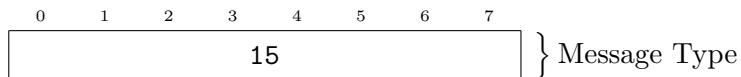


Abbildung 2.45.: Application-to-Service-Protokoll – Key Encryption Block Size Request Message

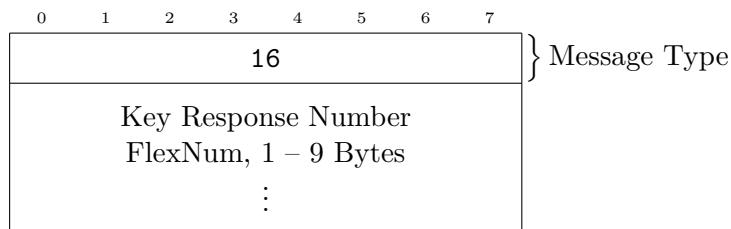


Abbildung 2.46.: Application-to-Service-Protokoll – Key Number Response Message

Anfrage, die eine Antwort durch eine Key Number Response Message erfordert, die Key Encryption Block Size Request Message.

2.10. Application Channels

2.10.1. Einleitung

Um Anwendungen, deren Kommunikation über den DCL geleitet werden soll, nicht nur das Senden von verbindungslosen Paketen zu ermöglichen, sondern auch die Übertragung von Daten durch verschlüsselte, performante und verlässliche Verbindungen, gibt es im DCL sogenannte Application Channels. Zur Zeit sind das direkte Kanäle zwischen jenen zwei DCL-Services, die mit den Anwendungen, zwischen denen der Application Channel besteht, via Application-to-Service-Verbindungen verbunden sind. Eine Implementierung eines zweiten Typs von Application Channels, bei dem die DCL-Services, die die Anwendungen an den DCL anbinden, nicht direkt miteinander verbunden sind, ist für die Zukunft nicht ausgeschlossen.

2.10.2. Sicherheit

Die durch einen Application Channel übertragenen Daten werden im Interservice Channel mit der Application Channel Data Message (2.8.7.14 Application Channel Data, Seite 35) zwischen den DCL-Services übermittelt. Der Link, über den der Interservice Channel übertragen wird, ist dabei verschlüsselt. Die Daten des Application Channel werden nicht weiter verschlüsselt. Da Application Channels aber immer zwischen genau zwei DCL-Services übertragen werden, ist neben der Verschlüsselung des Links auch keine weitere Verschlüsselung notwendig.

Selbstverständlich kann aber auf der Seite der Anwendung eine zusätzliche Verschlüsselungsschicht umgesetzt werden, sodass ausschließlich die verschlüsselten Daten über den Application Channel übertragen werden.

2. Decentralized Communication Layer

2.10.3. Aufbau

Für einen Application Channel müssen die beiden DCL-Services, die die Anwendungen an den DCL anbinden, direkt miteinander verbunden sein und über einen Interservice Channel miteinander kommunizieren. Beim Aufbau eines Application Channel gibt es deshalb zwei Fälle zu unterscheiden: der Fall, in dem die DCL-Services bereits miteinander verbunden sind und der Fall, in dem die DCL-Services noch nicht direkt miteinander verbunden sind.

Im zweiten Fall müssen die DCL-Services zusätzlich zu den Aktionen, die schon im ersten Fall notwendig sind, noch miteinander verbunden werden, wofür der Austausch ihrer LLAs und evtl. die Durchführung von NAT-Traversal notwenig sind.

Um diese Fälle in der Umsetzung nicht weiter behandeln zu müssen, werden die Anfragen für Application Channels über das Common Routed Interservice Protocol (CRISP) verschickt, welches unter 2.10.4 CRISP auf Seite 50 beschrieben wird.

Der DCL-Service, der die Anwendung an den DCL anbindet, die den Application Channel angefragt hat, sendet zum Aufbau eine CRISP Neighbor Request Message (siehe Abbildung 2.48, CRISP – Neighbor Request Message, Seite 51) über das Netzwerk, das die Anwendung in der Application Channel Outgoing Request Message über die Application-to-Service-Verbindung angegeben hat. Der Action Identifier aus der über die Application-to-Service-Verbindung gestellten Anfrage der Anwendung erhält das Präfix `org.dclayer.applicationchannel/` und wird in die Neighbor Request Message kopiert. Außerdem wird die LLA des lokalen DCL-Service in die Message übernommen, das Response-Flag auf `false` gesetzt und die Ignore Data eingefügt. Die Neighbor Request Message wird über das Netzwerk an die Adresse gesendet, die die Anwendung als Zieladresse des Application Channel angegeben hat.

Der entfernte DCL-Service, der die Neighbor Request Message empfängt, leitet die Anfrage per Application Channel Incoming Request Message über die Application-to-Service-Verbindung an die Anwendung weiter. Nimmt die Anwendung den Application Channel an, sendet diese als Antwort auf die Application Channel Incoming Request Message eine Application Channel Accept Message an den DCL-Service. Dieser antwortet danach wiederum dem ersten DCL-Service mit einer CRISP Neighbor Request Message, in der das Response-Flag auf `true` gesetzt wird, wodurch das Feld für die Ignore Data wegfällt. In die Message wird die LLA dieses DCL-Service eingetragen und der Action Identifier aus der vorhergehenden Neighbor Request Message kopiert. Sofern die beiden DCL-Services nicht bereits miteinander verbunden sind, sendet der DCL-Service neben der Neighbor Request Message gleichzeitig auch ein NAT-Traversal-Paket an die LLA des ersten DCL-Service, das aus der Ignore Data der zuvor empfangenen Message besteht, um ein möglicherweise notwendiges NAT-Traversal durchzuführen. Sollte kein NAT-Gerät die Durchführung von NAT-Traversal erfordern und dieses NAT-Traversal-Paket deshalb den anderen DCL-Service erreichen, so wird das Paket dort ignoriert, da es aus der zuvor übermittelten Ignore Data besteht. Für nähere Informationen zu NAT und NAT-Traversal siehe 2.6 NAT-Traversal, Seite 17.

Anschließend sendet der erste DCL-Service eine normale Verbindungsanfrage an die in der eben empfangenen Neighbor Request Message angeführte LLA, sofern die beiden DCL-Services nicht bereits verbunden sind. Die Anfrage erreicht aufgrund des zuvor durchgeführten NAT-Traversals jedenfalls ihr Ziel und die Verbindung wird aufgebaut.

Sobald die Verbindung aufgebaut ist und der Interservice Channel bereit ist, benachrichtigen die DCL-Services die Anwendungen durch Senden einer Application Channel Connected

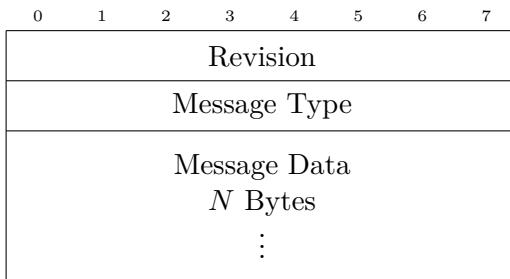


Abbildung 2.47.: CRISP – Genereller Messageaufbau

Message über die Application-to-Service-Verbindung darüber, dass der Application Channel verbunden wurde und Daten darüber übertragen werden können.

2.10.4. CRISP

Das Common Routed Interservice Protocol (CRISP) ist ein Protokoll, das zur verbindungslosen Kommunikation zwischen DCL-Services dient. Die Messages des Protokolls werden innerhalb des DCL durch Netzwerke geroutet. CRISP kann dabei unabhängig vom verwendeten Netzwerk genutzt werden.

Prinzipiell enthalten CRISP-Messages einen FlexNum-Component, der die Revisionsnummer des Protokolls angibt und einen weitere FlexNum-Component, der den Typ der Message angibt und anhand dessen der Rest der Message interpretiert wird.

Derzeit ist nur eine CRISP-Message definiert: die Neighbor Request Message.

Die Neighbor Request Message dient zur Anforderung einer direkten Verbindung zwischen zwei DCL-Services, die derzeit nur verbindungslos über ein Netzwerk kommunizieren können. Der primäre Verwendungszweck liegt derzeit im Aufbau von Application Channels, die Message kann aber auch zur Netzwerkintegration verwendet werden.

In der Neighbor Request Message wird ein Action Identifier übertragen, der den Grund für die Verbindungsanfrage enthält. Außerdem enthält die Message die LLA des Senders, ein Response-Flag, das angibt, ob es sich bei der Nachricht um eine Anfrage oder um eine Antwort handelt und Ignore Data, die für NAT-Traversal-Pakete verwendet werden können.

2.11. Packet Components

FlexNum

Ein FlexNum-Component ist ein Packet Component zur Übertragung einer bis zu 64 Bits langen Ganzzahl. Die Besonderheit liegt darin, dass der FlexNum-Component zwar Zahlen bis zu $2^{64} - 1$ übertragen kann, für Zahlen von 0 bis $2^7 - 1$, welche bereits mit 7 Bits darstellbar sind, trotzdem nur ein Byte zur Übertragung benötigt.

Das funktioniert durch Kodierung der Anzahl der folgenden Bytes im ersten Byte. Hierbei zeigt die Anzahl der führenden Bits mit dem Wert 1 im ersten Byte an, wie viele Bytes für die Darstellung der Zahl folgen. Ein Bit mit dem Wert 0 wird im ersten Byte also jedenfalls benötigt, um die Abfolge der führenden Bits mit dem Wert 1 zu beenden. Im Fall von Zahlen von 0 bis 127 ($2^7 - 1$) hat das erste Bit im ersten Byte den Wert 0 und

2. Decentralized Communication Layer

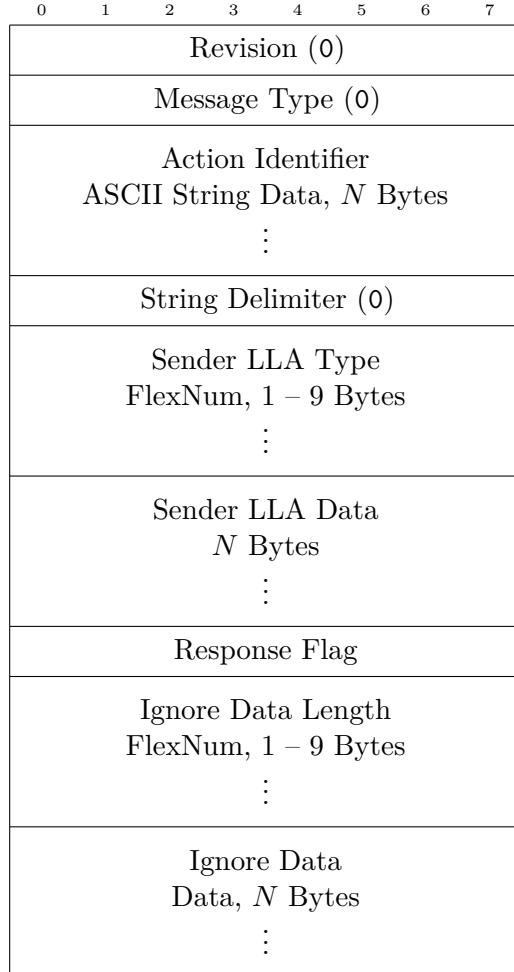


Abbildung 2.48.: CRISP – Neighbor Request Message

zeigt damit an, dass keine Bytes folgen. Die restlichen 7 Bits werden zur Darstellung der Zahl verwendet.

Nachfolgend ist eine Tabelle angeführt, die einen Überblick über die mit einem FlexNum-Component darstellbaren Wertbereiche, das Bitpräfix im Initialisierungsbyte, die Anzahl der folgenden Bytes sowie die Effizienz der Speichernutzung gibt.

Die Effizienz berechnet sich durch $\frac{\log_2(n_{max})}{N_{bytes} \cdot 8}$, wobei N_{bytes} der Gesamtanzahl an benötigten Bytes entspricht und n_{max} der mit der jeweiligen Gesamtanzahl an Bytes maximal darstellbare Wert ist.

Erstes Byte	Wertbereich	Folgebytes	Effizienz
0XXXXXXX	0 – 127	0	87,50%
10XXXXXX	128 – 16511	1	87,57%
110XXXXX	16512 – 2113663	2	87,55%
1110XXXX	2113664 – 270549119	3	87,54%
11110XXX	270549120 – 34630287487	4	87,53%
111110XX	34630287488 – 4432676798591	5	87,52%
1111110X	4432676798592 – 567382630219903	6	87,52%
11111110	567382630219904 – 72624976668147839	7	87,52%
11111111	72624976668147840 – 18519369050377699455	8	88,90%

Tabelle 2.1.: FlexNum – Darstellbare Werte

Nachfolgend ist jener Teil des Quellcodes von DCL angeführt, welcher zum Lesen eines FlexNum-Component angewandt wird.

```

1 long readFlexNum() {
2
3     byte iByte = read();
4     int additionalBytes = 8;
5     long num = 0;
6     long offset = 0;
7
8     for(int i = 0; i < 8; i++) {
9         if((iByte & (0x80 >> i)) == 0) {
10             additionalBytes = i;
11             break;
12         }
13     }
14
15     num |= (iByte & ((0x100 >> additionalBytes) - 1));
16     for(int i = 0; i < additionalBytes; i++) {
17         num <= 8;
18         num |= (read() & 0xFF);
19         offset += (0x80L << 7*i);
20     }
21 }
```

2. Decentralized Communication Layer

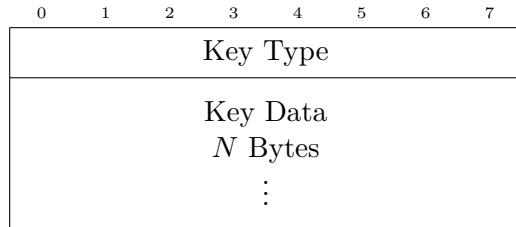


Abbildung 2.49.: KeyComponent – Genereller Componentaufbau

```
22     return num + offset;  
23 }  
24 }
```

Listing 2.3: Lesen eines FlexNum-Component (Java)

KeyComponent

Der KeyComponent dient zur Übertragung kryptographischer Schlüsselinformationen.

Dabei definiert der Packet Component einen grundsätzlichen Aufbau, bei dem ein einzelnes Byte am Beginn des Packet Component den Typ des Schlüssels angibt, anhand dessen die Daten eingelesen werden.

Derzeit kann der KeyComponent nur RSA-Schlüssel übertragen.

2.11.0.1. RSA Key

Der RSA Key Component überträgt den Modul und den Exponenten des RSA-Schlüssels.

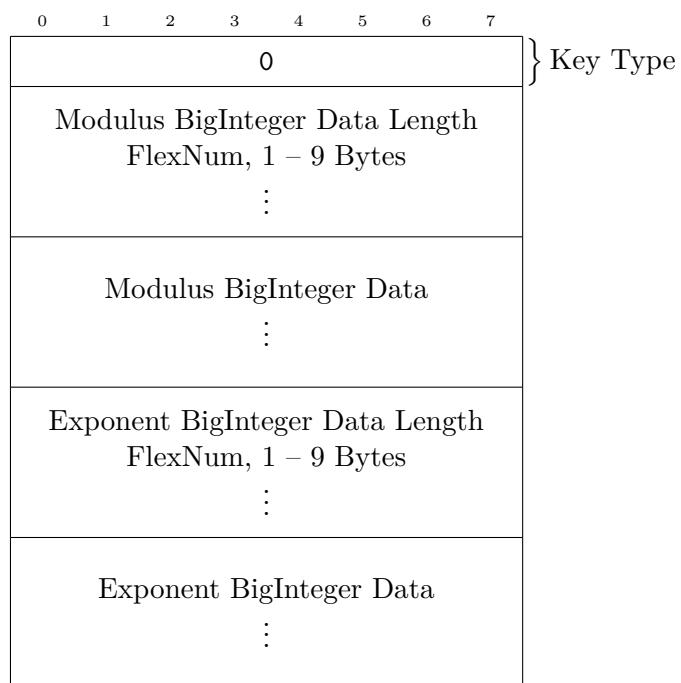


Abbildung 2.50.: KeyComponent – RSA Key Component

3. sblit

3.1. Einleitung

Die Anwendung sblit dient der Verwaltung des Ordners und aller Unterordner, die die zu synchronisierenden Dateien enthalten. Sie erkennt automatisch, wenn eine Datei bearbeitet wird, entscheidet, ob eine Datei synchronisiert werden soll und behebt etwaige Dateikonflikte.

3.2. Konfiguration

3.2.1. Allgemein

Mit dem Konfigurieren von sblit werden benutzerspezifische Anforderungen berücksichtigt und ein reibungsloser Betrieb sichergestellt. Die Konfiguration wird unter Windows im Appdata-Verzeichnis im Ordner *SBLIT* und auf Unix-basierten Systemen im Home-Verzeichnis im Ordner *.SBLIT* gespeichert. Des Weiteren wird das Verzeichnis nur für den Benutzer zugänglich gemacht. Dieses Verzeichnis sollte jedoch nicht direkt editiert werden. Sämtliche Einstellungen erfolgen über das Einstellungsmenü, das über den System-Tray erreicht werden kann.

Die Konfiguration enthält folgende Dateien:

- freceivers.txt
- logs.txt
- receivers.txt
- rk.txt
- sblitDirectory.txt
- symmetricKey.txt
- uk.txt

3.2.2. freceivers.txt

freceivers.txt enthält eine Liste der Partnergeräte. Darin wird festgehalten, ob die eigenen Dateien auf Partnergeräten gespeichert werden. Eine detaillierte Beschreibung findet sich im Kapitel 3.5 Partnerschaften, Seite 79.

Die Adress-Bytes der Partnergeräte werden durch Beistriche getrennt in hexadezimaler Form gespeichert.

Zum Schreiben der Datei wird folgender Code verwendet:

```

1 private synchronized void updateDevices(File file, Map<String
2   , String> devices) throws IOException {
3   String temp = devices.toString();
4   temp = temp.substring(1, temp.length() - 1);
5   Files.write(file.toPath(), temp.getBytes(),
6             StandardOpenOption.TRUNCATE_EXISTING);
}

```

Listing 3.1: Schreiben der Gerätedatei freceivers.txt

file

Der Parameter `file` legt die Datei fest, die die Adressen der Partnergeräte enthält. Diese Parametrisierung ist deswegen notwendig, weil die Methode `updateDevices(file, devices)` auch für das Schreiben von `receivers.txt` (siehe 3.2.4 `receivers.txt`, Seite 58) verwendet wird.

devices

Der Parameter `devices` enthält die Liste aller Geräte, auf denen die Daten gespeichert werden.

IOException

`IOException` weist auf einen Fehler beim Schreiben der Datei hin.

temp

`temp` enthält den Inhalt, der in die Datei geschrieben werden soll.

In den Zeilen 3 und 4 werden die Adressen der Partnergeräte in das Speicherformat übertragen. In Zeile 4 werden die durch die `.toString()`-Methode eingefügten geschwungenen Klammern wieder entfernt. In Zeile 5 wird die im `file`-Parameter angegebene Datei mit den neuen Adressen aktualisiert. `StandardOpenOption.TRUNCATE_EXISTING` bedeutet, dass die Datei `freceivers.txt` überschrieben wird, falls diese schon vorhanden ist.

3.2.3. logs.txt

Das Logfile (siehe 3.3.2 Versionierung, Seite 59) `logs.txt` enthält die Versionsverläufe aller Dateien im sbplit-Ordner und dessen Unterordnern im Format:

`Pfad1=Version1,Version2;Adresse1,Adresse2;Pfad2`

`Pfad1` bezeichnet den relativen Pfad vom sbplit-Ordner zur betrachteten *Datei 1*. Die einzelnen Versionen (`Version1, Version2`) sind Hashwerte der *Datei 1* zu verschiedenen Zeitpunkten, deren Bytes in hexadezimaler Form im Logfile gespeichert werden. Analog dazu werden auch die Adressen der Geräte, die bereits die aktuellste Version der *Datei 1* enthalten, in hexadezimaler Form gespeichert. `Pfad2` ist der Pfad zu jener Datei, mit der *Datei 1* in Konflikt steht. Dieser ist optional.

Zum Schreiben des Logfiles wird folgende Methode verwendet:

```

1 private synchronized void write(Map<String, LinkedList<Data>>
2     versions, Map<String, LinkedList<Data>>
3     synchronizedDevices, Map<String, String> conflictOf)
4     throws IOException {
5     String logs = "";
6     if (versions.size() > 0) {
7         StringBuilder builder = new StringBuilder();
8         for (String path : versions.keySet()) {
9             if (new File(path).isFile()) {
10                 builder.append("\n");
11                 builder.append(path);
12                 builder.append("=");
13                 StringBuilder temp = new StringBuilder();
14                 for (Data data : versions.get(path)) {
15                     temp.append(",");
16                     temp.append(data.toString());
17                 }
18                 temp.append(";");
19                 //substring(1) wird verwendet, damit der
20                 //erste Beistrich entfernt wird.
21                 builder.append(temp.substring(1));
22                 temp = new StringBuilder();
23                 for (Data data : synchronizedDevices.get(path))
24                     {
25                         temp.append(",");
26                         temp.append(data.toString());
27                     }
28                 if(conflictOf.get(path) != null) {
29                     temp.append(";");
30                     temp.append(conflictOf.get(path));
31                 }
32                 //substring(1) wird verwendet, damit der erste
33                 //Zeilenumbruch entfernt wird.
34                 s = builder.substring(1);
35             }
36             Files.write(logFile.toPath(), s.getBytes());
37         }
38     }
39 }
```

Listing 3.2: Schreiben des Logfiles *logs.txt*

versions

versions enthält alle im sblit-Ordner vorhandenen Dateien und deren Versionsverlauf.

synchronizedDevices

synchronizedDevices enthält alle im sblit-Ordner vorhandenen Dateien und die Geräte, auf denen die aktuellste Version bereits gespeichert ist.

logs

logs enthält die formatierten Versionsverläufe.

temp

Diese Variable enthält einen bestimmten Versionsverlauf, der gerade bearbeitet wird.

Die Methode **write** überprüft zunächst, ob Dateien in der Versionsliste **version** vorhanden sind. Anschließend werden die Dateien in ein Format gebracht, in dem sie gemeinsam mit dem Datei- und Versionsverlauf in die Datei geschrieben werden können. Falls eine Datei die Konfliktdatei einer anderen ist, wird diese ebenfalls in die Datei geschrieben.

3.2.4. receivers.txt

receivers.txt enthält eine Liste der Geräte, auf denen der sblit-Ordner synchronisiert werden soll. Zu diesen Geräten wird optional ein Name gespeichert, um es dem Benutzer zu erleichtern, den Überblick über die Geräte zu bewahren. Diese werden in der Form **Adresse=Name** gespeichert, wobei die Adresse wieder eine hexadezimale Darstellung der Bytes ist und der Name vom Nutzer beliebig vergeben werden kann.

3.2.5. rk.txt

rk.txt enthält den Private-Key des Gerätes. Dieser dient dazu, den Besitz Adresse zu beweisen und ist unbedingt geheim zu halten, da sonst fremde Geräte Identitätsklau begehen könnten. Der Private-Key wird in hexadezimaler Form in der Datei gespeichert. Die Datei ist außerdem versteckt.

3.2.6. sblitDirectory.txt

sblitDirectory.txt enthält den Pfad zum sblit-Ordner. Darin wird ein **String** gespeichert, der den Dateipfad enthält.

3.2.7. symmetricKey.txt

In *symmetricKey.txt* wird der symmetrische Schlüssel (siehe ?? ??, Seite ??) festgehalten. Dieser dient zum Entschlüsseln der Daten, die auf den Partnergeräten gespeichert werden. Der symmetrische Schlüssel wird, wie alle binären Daten, in hexadezimaler Form in der Datei gespeichert. Zusätzlich ist diese Datei versteckt.

3.2.8. uk.txt

uk.txt enthält den Public-Key des Gerätes, der gleichzeitig die Geräteadresse ist. Beim Verbinden mit DCL wird diese Adresse bekanntgegeben. Der Schlüssel wird in hexadezimaler Form in der Datei gespeichert.

3.3. Datei-Verarbeitung

3.3.1. Allgemein

Ein wesentlicher Bestandteil von sblit ist die Verarbeitung der Dateien. Dazu zählen das Erkennen von Änderungen sowie das Erkennen und Lösen von Konflikten.

3.3.2. Versionierung

Ein wichtiges Mittel zur Verwaltung der Dateien ist das Logfile. Dieses enthält Informationen zu allen Dateien im sblit-Ordner. Dazu zählen der relative Dateipfad, eine Liste mit Hashes, die für die Versionierung zuständig sind, und eine Liste an Geräten, auf denen die Datei bereits auf dem aktuellen Stand ist. Weiters wird in dieser Datei festgehalten, ob Dateien im Konflikt zu anderen Dateien stehen. Die Versionierung ist vor allem für die Konflikterkennung (siehe 3.3.5.2 Konflikterkennung, Seite 64) notwendig. Hierbei werden Hashes für alle Versionen der Dateien gespeichert. Die Version wird beim Speichern der Datei um den aktuellen Hashwert erweitert und bei Konvergenz auf allen Geräten auf die aktuelle gemeinsame Version reduziert.

3.3.3. Löschen auf Partnergeräten

Damit sblit weiß, wann die Dateien von den Partnergeräten (siehe 3.5.2 Partnergeräte, Seite 79) wieder gelöscht werden können, wird eine Liste von Geräten, mit Dateien auf dem aktuellen Stand, gespeichert. Diese Liste wird bei jeder Änderung mit der Liste aller eigenen Geräte verglichen. Sobald alle eigenen Geräte eingetragen sind, werden die Partnergeräte aufgefordert, die Datei zu löschen.

3.3.4. Reaktionen auf Dateiänderungen

Wenn eine Datei neu erstellt, bearbeitet oder gelöscht wird, erkennt dies sblit und leitet diese Informationen an den Synchronisationsprozess weiter. Außerdem ist das Erkennen einer Änderung im sblit-Ordner wichtig, damit sie im Logfile protokolliert werden kann. Das Logfile wird sowohl für die Konflikterkennung (siehe 3.3.5.2 Konflikterkennung, Seite 64) als auch für die Reduktion der benötigten Bandbreite (siehe 3.4.3.2 File Request, Seite 68) genutzt.

Um Änderungen zu erkennen, verwendet sblit das sogenannte WatchService. Das WatchService wird bei Änderungen in dem Ordner, der vom Benutzer für die Synchronisation festgelegt wurde, benachrichtigt. Je nach dem, ob eine Datei angelegt, verändert oder gelöscht wurde, erfolgen unterschiedliche Arbeitsschritte. Beim Anlegen einer neuen Datei wird ihr Pfad samt Hash in das Logfile geschrieben und eine Dateianfrage an die anderen eigenen Geräte geschickt. Wird eine Datei verändert, wird ein neuer Hash zu den bereits vorhandenen Hashes hinzugefügt. Außerdem wird eine Dateianfrage an die anderen eigenen Geräte gesendet. Wird eine Datei gelöscht, wird sie samt Hashes aus dem Logfile entfernt. Anschließend wird eine Löschanfrage an die eigenen Geräte versandt. Um zu erkennen, ob die Änderung vom Benutzer oder vom Programm vorgenommen wurde, überprüft sblit zuerst das Logfile. Steht dort zum Hash der aktuellen Version schon ein anderes Gerät, wird keine Anfrage an dieses Gerät geschickt.

```

1 WatchService watcher = filesDirectory.getFileSystem()
2     .newWatchService();
3 filesDirectory.register(watcher,
4     StandardWatchEventKinds.ENTRY_CREATE,
5     StandardWatchEventKinds.ENTRY_DELETE,
6     StandardWatchEventKinds.ENTRY_MODIFY);
7 WatchKey watchKey = watcher.take();
8 List<WatchEvent<?>> events = watchKey.pollEvents();
9 Map<String, LinkedList<Data>> logs = getLogs();
10 Map<String, LinkedList<Data>> synchronizedDevices =
11     getSynchronizedDevices();

```

Listing 3.3: Initialisierung des WatchService

filesDirectory

`filesDirectory` enthält den vom Benutzer festgelegten sbplit-Ordner.

watcher

`watcher` wird benötigt, um das WatchService einen bestimmten Ordner überwachen zu lassen.

watchKey

`WatchKey` enthält eine Liste an Dateien, die neu erstellt, geändert oder gelöscht wurden.

events

Jedes `WatchEvent` aus der `List events` enthält eine Datei, die sich geändert hat, und die Information, ob sie neu erstellt, verändert oder gelöscht wurde.

logs

`logs` enthält den Versionsverlauf aller Dateien, die sich im sbplit-Ordner befinden.

synchronizedDevices

`synchronizedDevices` enthält zu jeder Datei eine Liste an Geräten, auf denen die aktuelle Version gespeichert ist.

Zunächst wird ein neues Objekt vom Typ `WatchService` erstellt. Dieses überwacht den festgelegten sbplit-Ordner hinsichtlich neu erstellter, geänderter und gelöschter Dateien. Zur Registrierung des WatchService im Dateisystem, werden in der `register`-Methode das WatchService-Objekt und die Fälle, in denen das WatchService benachrichtigt werden soll, festgelegt. In diesem Fall sind das neue Dateien (`ENTRY_CREATE`), Dateiänderungen (`ENTRY MODIFY`) und gelöschte Dateien (`ENTRY_DELETE`). Die `take`-Methode wartet auf Änderungen im festgelegten Ordner und übergibt sie dann an das `WatchKey`-Objekt. Die Anweisung `watchKey.pollEvents()` unterteilt den WatchKey in einzelne Dateiänderungen und formatiert diese für den weiteren Gebrauch.

Der Thread wird angehalten, da das Programm beim Empfangen einer Datei das Logfile bearbeitet. Dies passiert, damit sbplit weiß, dass die Datei nicht vom Nutzer verändert wurde.

3. sblit

```
1 for (WatchEvent<? extends File> event : events) {  
2     String path = event.context().toString();  
3     if(!(path.startsWith("\\\\sblit\\\\.") || path.contains(  
4         Configuration.slash + "\\\\sblit\\\\."))){  
      File changedFile = new File(Configuration.  
          getSblitDirectory() + Configuration.slash + path);
```

Listing 3.4: Unterteilen in einzelne Dateien

event

event enthält Informationen zur Erstellung, Änderung oder Bearbeitung einer Datei aus der Liste events.

path

path enthält den relativen Pfad zur geänderten Datei.

changedFile

changedFile enthält die Datei, die gerade geändert wurde.

Neben dem Unterteilen der Dateien wird überprüft, ob die Datei mit (.sblit.) anfängt. Ist dies der Fall, wird die Datei nicht synchronisiert, da es sich dabei um eine unvollständig synchronisierte Datei handelt.

```
1 if (event.kind() == StandardWatchEventKinds.ENTRY_CREATE &&  
    logs.get(path) == null) {  
    byte[] fileContent = Files.readAllBytes(changedFile.  
        toPath());  
    Data hash = Crypto.sha1(new Data(fileContent));  
    LinkedList<Data> hashes = new LinkedList<>();  
    hashes.add(hash);  
    logs.put(path, hashes);  
    LinkedList<Data> devices = new LinkedList<>();  
    devices.add(Configuration.getPublicAddressKey().toData())  
        ;  
    synchronizedDevices.put(path, devices);  
    filesToPush = refreshFilesArray(filesToPush, changedFile)  
        ;  
}
```

Listing 3.5: Erstellen einer Datei

fileContent

fileContent enthält den Inhalt der Datei, die neu erstellt wurde.

hash

In hash wird der Hashwert des Inhalts der Datei gespeichert.

hashes

hashes enthält den Versionsverlauf einer Datei.

devices

devices enthält alle Geräte, auf denen die Datei aktuell ist. In diesem Fall ist das genau das Gerät, auf dem der Code ausgeführt wird, da diese Datei gerade erst erstellt wurde.

Wenn eine Datei erstellt wurde, wird sie zu allererst ausgelesen. Anschließend wird der Inhalt verhasht und dem Versionverlauf hinzugefügt. Außerdem wird das Gerät, auf dem der Code ausgeführt wird, zur Geräteliste **devices** hinzugefügt. Schließlich wird die Datei zur Liste der zu synchronisierenden Dateien hinzugefügt.

```

1 else if (event.kind() == StandardWatchEventKinds
2     .ENTRY_DELETE) {
3     logs.remove(path);
4     synchronizedDevices.remove(path);
5     filesToDelete = refreshFilesArray(filesToDelete,
6         changedFile);
7 }
```

Listing 3.6: Löschen einer Datei

Beim Löschen einer Datei wird der zugehörige Versionsverlauf ebenfalls entfernt. Außerdem wird der Variable **filesToDelete** der Pfad der gelöschten Datei hinzugefügt, um die anderen eigenen Geräte auch darüber zu informieren, dass eine Datei gelöscht wurde.

```

1 else if (event.kind() == StandardWatchEventKinds
2     .ENTRY MODIFY) {
3     byte[] fileContent = readFile(event);
4     Data hash = Crypto.sha1(new Data(fileContent));
5     LinkedList<Data> hashes = logs.get(path);
6     if (!hashes.contains(hash)) {
7         hashes.add(hash);
8         filesToPush = refreshFilesArray(filesToPush,
9             changedFile);
10    LinkedList<Data> devices = new LinkedList<>();
11    devices.add(Configuration
12        .getPublicKey().toData());
13    synchronizedDevices.put(path, devices);
14 }
15 }
```

Listing 3.7: Bearbeiten einer Datei

fileContent

fileContent enthält den Inhalt der Datei, die bearbeitet wurde.

hash

In **hash** wird der Hashwert des Inhalts der Datei gespeichert.

hashes

`hashes` enthält den Versionsverlauf der Datei.

devices

`devices` enthält alle Geräte, auf denen die Datei aktuell ist. In diesem Fall ist das genau jenes Gerät, auf dem der Code ausgeführt wird, da diese Datei gerade bearbeitet wurde.

Wurde die Datei bearbeitet, wird sie ausgelesen und anschließend verhasht. Nachdem der Versionsverlauf für die Datei in der Variable `hashes` gespeichert wurde, wird überprüft, ob der Versionsverlauf die Datei schon enthält. Ist dies der Fall, ist ein Fehler aufgetreten und die neue Version wird nicht synchronisiert. Falls der Versionsverlauf die aktuelle Version noch nicht enthält, wird der Hash der aktuellen Version im Versionsverlauf gespeichert. Anschließend wird eine neue Liste an Geräten erstellt, auf denen schon die neueste Version synchronisiert ist. In diesem Fall wird nur das eigene Gerät hinzugefügt, da die neueste Version bis jetzt nur auf diesem Gerät vorhanden ist.

```
1     }
2 }
3 logFile.createNewFile();
4 write(files, synchronizedDevices);
5 writeFilesToDelete(filesToDelete);
```

Listing 3.8: Schreiben des Logfiles

Das Logfile wird neu erstellt und die neuen Daten werden anschließend darin gespeichert. Außerdem werden die zu löschen Dateien in eine Datei geschrieben, damit sie auch auf Geräten gelöscht werden können, die gerade nicht online sind.

3.3.5. Konflikte

3.3.5.1. Allgemein

Ein Konflikt ist ein Problem, das bei Synchronisationsdiensten vorkommt. Dieser tritt auf, wenn eine Datei bearbeitet wird, bevor sie synchronisiert werden kann.

Dazu ein Beispiel: Susanne hat einen Laptop und einen Stand-Rechner, auf denen sie mithilfe von sblit einen Ordner synchronisiert. Im Normalfall, also wenn kein Konflikt auftritt, bearbeitet Susanne eine Datei auf dem Laptop. Nach dem Speichern wird die Datei auf den Stand-Rechner übertragen und dort gespeichert. Die Datei ist nun auf beiden Geräten synchron.

Angenommen jedoch, Susanne schaltet nun den Laptop aus und bearbeitet anschließend die Datei noch einmal auf dem Stand-Rechner. Da der Laptop ausgeschaltet ist, kann die Datei nicht synchronisiert werden. Auf dem Weg zur Arbeit fällt Susanne noch eine Verbesserungsmöglichkeit der Datei ein und sie bearbeitet die Datei auf dem Laptop ohne einer Verbindung zum Internet. In der Arbeit angekommen, packt Susanne wieder ihren Laptop aus und verbindet sich zum Internet. Die Datei kann jetzt nicht auf den neuesten Stand gebracht werden, da ja zwei unterschiedliche Versionen existieren. Würde die Datei einfach vom Stand-Rechner auf den Laptop kopiert werden, gingen die Neuerungen am

Laptop verloren, umgekehrt würde die Datei vom Laptop die Änderungen am Stand-Rechner überschreiben. Diesen Zustand zweier verschiedener Versionen der gleichen Datei, ohne die Änderungen des jeweils anderen Gerätes schon berücksichtigt zu haben, nennt man einen Konflikt.

3.3.5.2. Konflikterkennung

Um Konflikte zu erkennen, verwendet sblit eine interne Versionierung der Dateien. Diese wird im Kapitel 3.3.2 Versionierung, Seite 59 näher erklärt. Bei einer Dateianfrage (siehe 3.4.3.2 File Request, Seite 68) werden alle Hashes einer Datei mitgesendet. Diese werden nach dem Empfang der Anfrage mit den Hashes der Datei mit dem gleichen Namen und den Hashes aller Konfliktdateien dieser Datei im lokalen Logfile verglichen. Stimmen die beiden aktuellen Hashes aus Dateianfrage und der Datei mit dem gleichen Namen im Logfile überein, muss die Datei gar nicht übertragen werden, da sie schon auf dem neuesten Stand ist. Das heißt natürlich auch, dass kein Konflikt auftritt. Stimmt der aktuelle Hash im lokalen Logfile mit einem in der Dateianfrage überein, tritt ebenfalls kein Konflikt auf, da der aktuelle lokale Hash dem Gerät, das die Anfrage verschickt hat, schon bekannt ist.

Stimmt der Versionsverlauf aus dem File Request mit dem einer Konfliktdatei überein, wurde eine Version synchronisiert, bevor ein drittes Gerät einen Konflikt bemerkt hat. In diesem Falle könnten mehrere Konfliktdateien mit nur zwei verschiedenen Inhalten erstellt werden. Passiert dies, werden die Namen der beiden Dateien vertauscht, um das eben genannte Szenario zu verhindern.

Ist der aktuelle Hash der lokalen Datei jedoch nicht in der Dateianfrage enthalten, wurde die lokale Datei bearbeitet, bevor diese auf den aktuellsten Stand gebracht werden konnte. Anders gesagt: Ein Konflikt ist aufgetreten.

3.3.5.3. Konfliktlösung

Damit die Änderungen von einem Gerät nicht überschrieben werden, speichert sblit beide Versionen. Da jedoch beide Dateien nicht den gleichen Namen haben können, benennt das Gerät, das den Konflikt erkennt, (im Folgenden Gerät A) die lokale Datei um. Die Datei wird folgendermaßen umbenannt: *Datei.Endung* wird zu *Datei (Konflikt [Konfliktzahl]).Endung*.

Anschließend schickt Gerät A eine Antwort an das Gerät, das die Dateianfrage geschickt hat, (im Folgenden Gerät B) in der es die neue Version anfordert, als ob kein Konflikt aufgetreten wäre. Außerdem schickt Gerät A eine Dateianfrage mit der neuen Datei an Gerät B. Gerät B sendet nun die angeforderte Datei an Gerät A und akzeptiert die Konfliktdatei, da es eine Datei mit dem gleichen Namen noch nicht besitzt.

Nach dem Empfang der Datei speichert Gerät A diese unter dem ursprünglichen Namen. Des Weiteren sendet Gerät A die Konfliktdatei mit dem geänderten Namen an Gerät B. Die neue Datei wird auf Gerät B unter dem neuen Namen gespeichert.

3.3.5.4. Umsetzung

Um Konflikte zu erkennen und zu lösen, wird der folgende Code verwendet:

3. sblit

```
1 private void handleConflict(LinkedList<Data> requestedHashes,
2     LinkedList<Data> ownHashes, String path) throws
3     IOException {
4     String sblitDirectory = Configuration.getSblitDirectory()
5         + Configuration.slash;
6     if (!requestedHashes.contains(ownHashes.get(ownHashes.
7         size() - 1))) {
8         int dotIndex = path.lastIndexOf(".");
9         File conflictFile;
10        if (dotIndex > path.lastIndexOf(Configuration.slash)
11            + 1) {
12            for (int i = 1;; i++) {
13                conflictFile = new File(sblitDirectory + path
14                    .substring(0, dotIndex) + "(Conflict " + i
15                    + ")" + path.substring(dotIndex));
16                if (!conflictFile.exists()) {
17                    break;
18                }
19            }
20        } else {
21            for (int i = 1;; i++) {
22                conflictFile = new File(sblitDirectory + path
23                    + "(Conflict " + i + ")");
24                if (!conflictFile.exists()) {
25                    break;
26                }
27            }
28        }
29        File file = new File(sblitDirectory + path);
30        Files.copy(file.toPath(), conflictFile.toPath());
31    }
32}
```

Listing 3.9: Erkennen eines Konflikts

requestedHashes

Der Parameter gibt an, welche Hashes in der Dateianfrage geschickt wurden. Die Hashes haben den Datentyp **Data**, welcher in DCL definiert wurde. Um eine unbestimmte Menge davon speichern zu können, werden diese in eine **LinkedList** geschrieben. Diese hat den Vorteil, dass die Elemente die Reihenfolge behalten, in der sie in die Liste geschrieben wurden.

ownHashes

Dieser Parameter enthält eine **LinkedList** an Hashes vom Datentyp **Data**, welche gemeinsam den Versionsverlauf der lokalen Datei ergeben.

path

Der Parameter path enthält den zum sblit-Ordner relativen Pfad.

sblitDirectory

Diese Variable vom Datentyp String enthält den absoluten Pfad des sblit-Ordners. Dieser setzt sich zusammen aus Configuration.getSblitDirectory() und Configuration.slash. Configuration.getSblitDirectory() liefert den sblit-Ordner zurück, der in der Configuration-Klasse definiert ist. Außerdem wird, je nach dem, ob das Betriebssystem Windows oder Unix-basierend ist, ein Backslash oder ein Slash an den Pfad gehängt. Die Zuweisung des richtigen Slashes wird am Programmstart bei der Initialisierung der Configuration-Klasse vorgenommen.

dotIndex

In dieser Variable befindet sich der Index des letzten Punktes im Dateipfad. Dieser dient dazu, herauszufinden, ob die Datei eine Endung hat.

conflictFile

Das conflictFile ist die neue Datei, die erzeugt wird, wenn ein Konflikt auftritt.

file

Die Variable file enthält die ursprüngliche Datei.

Enthält der empfangene Versionsverlauf nicht den aktuellsten Hash im Logfile, tritt ein Konflikt auf. Ist dies der Fall, wird zunächst überprüft, ob die Datei eine Dateiendung besitzt.

Hat die Datei eine Dateiendung, wird die Anmerkung, dass es sich um eine Konfliktdatei handelt, zwischen Dateinamen und Dateiendung eingefügt. Hat die Datei jedoch keine Dateiendung, wird die Anmerkung einfach zum Schluss eingefügt.

Existiert schon eine Datei mit dem gleichen Namen, den die Konfliktdatei bekommen soll, wird die Konfliktzahl so lange erhöht, bis eine Datei mit einem solchen Namen noch nicht existiert. Die Datei wird anschließend mit dem gewählten Dateinamen gespeichert.

3.4. Kommunikation

3.4.1. Allgemein

Um zu verhindern, dass Daten von Unbefugten mitgelesen werden, verwendet sblit den sicheren Application Channel (siehe Seite 48) der Kommunikationsschicht.

Dabei unterscheidet man grundsätzlich zwischen Nachrichten, die an Partnergeräte (siehe 3.5.2 Partnergeräte, Seite 79), und Nachrichten, die an eigene Geräte geschickt werden. Weiters gibt es Nachrichten, die bei beiden Arten von Geräten gleich sind.

3.4.2. Nachrichten an alle Geräte

3.4.2.1. Allgemein

Bevor ein fremdes Gerät eine Verbindung aufbaut, muss auf jeden Fall seine Identität überprüft werden. Dabei werden folgende Nachrichten versendet:

- Authenticity Requests

3. sblit

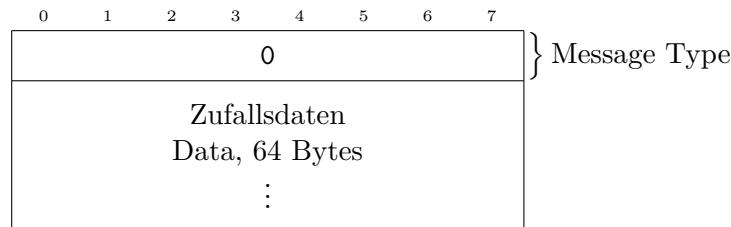


Abbildung 3.1.: sblit – Authenticity Request Message

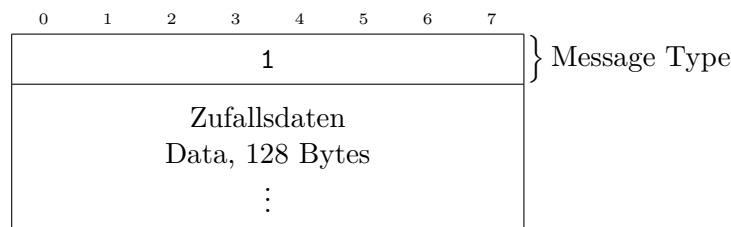


Abbildung 3.2.: sblit – Authenticity Response Message

- Authenticity Responses

3.4.2.2. Authenticity Request

Sobald zwei Geräte (Gerät A, Gerät B) Daten austauschen wollen, müssen sie die Identität des jeweiligen Kommunikationspartners überprüfen. Dies erfolgt über Authenticity Requests. Dabei schickt Gerät B zufällige Daten an Gerät A mit der Aufforderung, diese zu verschlüsseln (siehe Abbildung 3.1, sblit – Authenticity Request Message). Die Gesamtlänge der Daten, die mit RSA-2048 verschlüsselt werden, darf maximal 128 Byte betragen. Um zu verhindern, dass Gerät B bestimmte Daten mit dem Private-Key von Gerät A verschlüsseln lässt, werden 64 von den maximal 128 Byte reserviert. Diese 64 Byte nutzt Gerät A, um zufällige Daten hinzuzufügen, damit sich Gerät B nicht gewünschte Werte verschlüsseln lassen kann.

3.4.2.3. Authenticity Response

Bevor die Daten verschlüsselt werden, wird ein zufälliger Wert mit einer Länge von 64 Bit an die empfangenen Daten angefügt. Wird dieses Paket nun von Gerät B empfangen, kann der Inhalt mit dem Public-Key des Gerätes A, also dessen Adresse, entschlüsselt werden (siehe Abbildung 3.2, sblit – Authenticity Response Message).

3.4.3. Nachrichten an eigene Geräte

3.4.3.1. Allgemein

sblit verwendet zur Kommunikation zwischen den authentifizierten eigenen Geräten vier verschiedene Nachrichten:

- File Requests

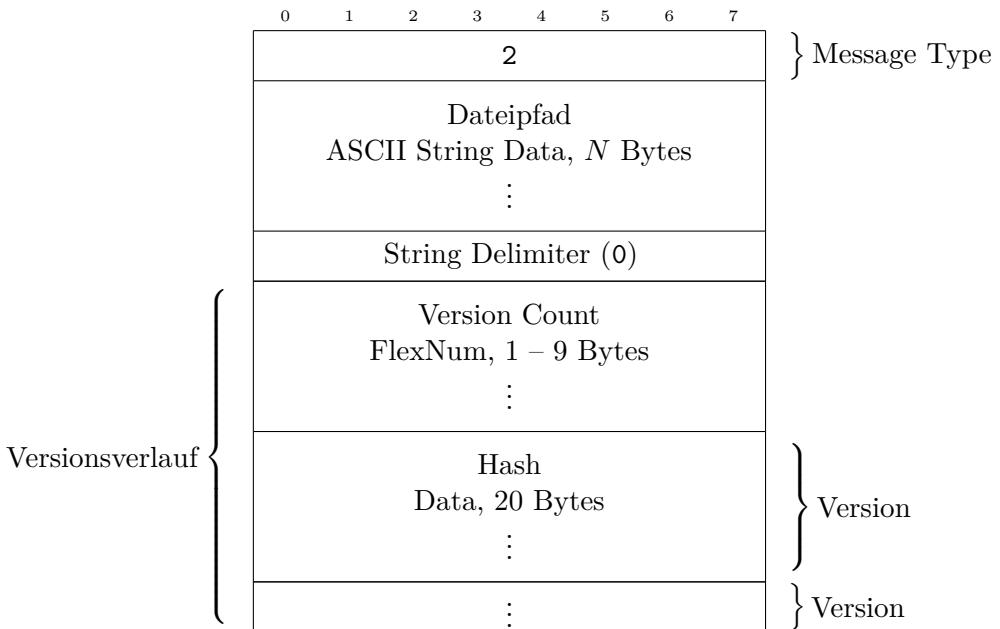


Abbildung 3.3.: sblit – File Request Message

- File Responses
- File Messages
- File Delete Messages
- Device Refresh Messages

3.4.3.2. File Request

Mithilfe des File Request kann eruiert werden, ob die Datei bereits auf einem anderen Gerät bereits vorhanden ist. Außerdem kann mit dem File Request ein möglicher Konflikt (siehe 3.3.5 Konflikte, Seite 63) ausgeschlossen werden (siehe Abbildung 3.3, sblit – File Request Message).

Dateipfad

Hierbei wird der zu sblit's Hauptordner relative Dateipfad mitgeschickt. Der absolute Dateipfad wird nicht mitgeschickt, da der Ort des sblit-Ordners nicht auf allen Geräten gleich sein muss. Befindet sich der Ordner auf Gerät A beispielsweise unter `C:/Users/Susanne/` kann sich der Ordner auf Gerät B auch unter `/home/susanne/dateien/` befinden.

Versionsverlauf

Der Versionsverlauf beinhaltet alle Hashes einer Datei seit der letzten komplett synchronisierten Version. Das heißt, dass auf jedem Gerät aktuell entweder diese oder eine neuere Version gespeichert ist.

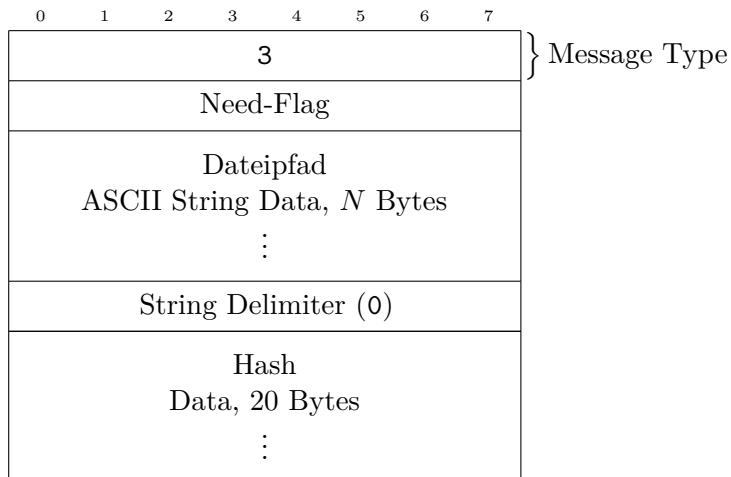


Abbildung 3.4.: sblit – File Response Message

3.4.3.3. File Response

Der File Response wird benötigt, um dem Gerät, das den File Request geschickt hat, zu antworten, ob die Datei benötigt wird.

Need-Flag

Dieses Feld enthält einen Hexadezimalwert, der darüber Auskunft gibt, ob die Datei benötigt wird oder nicht. Steht in diesem Feld der Hexadezimalwert 0x00, wird die Datei nicht benötigt, d.h., die aktuellste Version der Datei ist auf dem Gerät bereits vorhanden. Steht hier hingegen der Hexadezimalwert 0x01, wird die Datei benötigt. Dieses Byte hilft den Datenverkehr zu reduzieren. So muss nicht eine ganze Datei verschickt werden, obwohl diese gar nicht gebraucht wird.

Dateipfad

Wie auch beim File Request wird im File Response der zu sblit's Hauptordner relative Pfad mitgeschickt.

Hash

Hier wird noch einmal der letzte Hash des Versionsverlaufs der Dateianfrage verschickt, um sicherzustellen, dass die Datei in der Zwischenzeit nicht geändert wurde.

3.4.3.4. File Message

Die File Message dient zur Übertragung der Datei (siehe Abbildung 3.5, sblit – File Message).

Dateiinhalt

Der Dateiinhalt wird als binäres Data-Objekt verschickt.

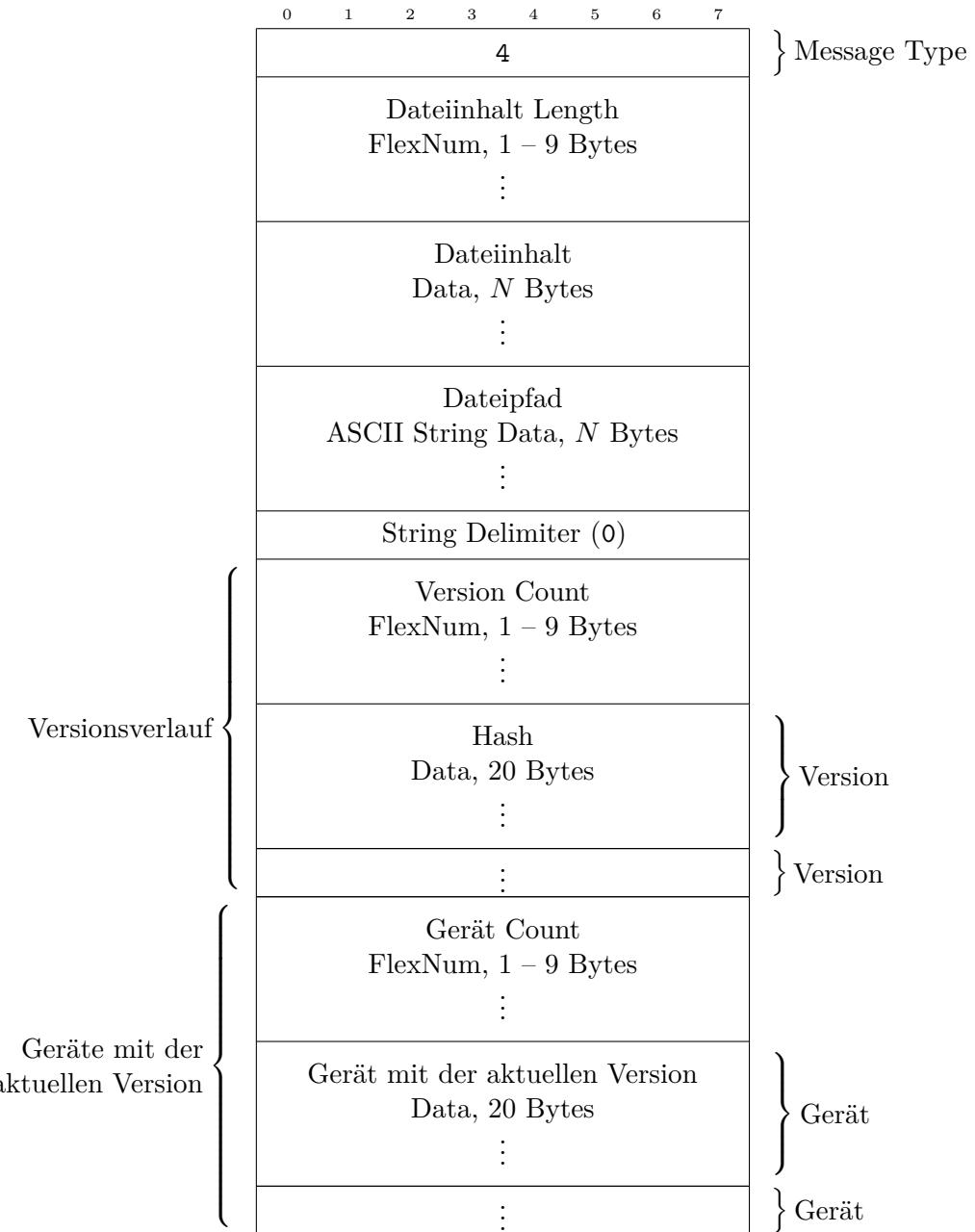


Abbildung 3.5.: sblit – File Message

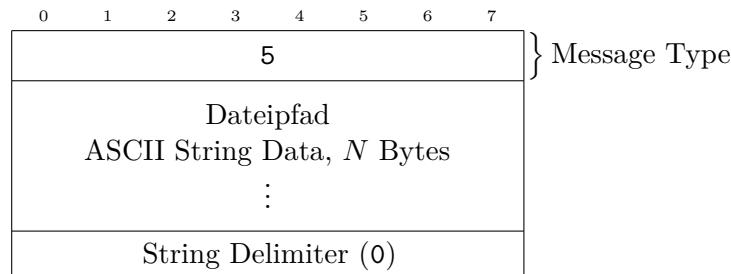


Abbildung 3.6.: sblit – File Delete Message

Geräte mit der aktuellen Version

Hier stehen die Adressen aller Geräte, die schon die neueste Version schon haben. Ist die Version auf allen Geräten aktuell, kann sie von den Partnergeräten gelöscht werden. Daher wird diese Liste an Geräten immer mit der Datei mitgeschickt. Außerdem können somit unnötige Anfragen an Geräte, die die Datei schon besitzen, verhindert werden.

Dateipfad

Der Dateipfad wird benötigt, damit das Gerät weiß, an welchem Ort die zu synchronisierende Datei gespeichert werden soll. Dies ist der gleiche relative Ort, wie auf dem Gerät, das die Anfrage geschickt hat.

Versionsverlauf

Der Versionsverlauf wird mitgeschickt, damit er auf allen Geräten einheitlich ist. Dies verhindert, dass Konflikte fälschlicherweise erkannt werden, wo keine vorhanden sind. Weiters stellt der aktuellste Hash sicher, dass die versendete Datei korrekt zugestellt wurde. Verhasht man die versandte Datei, muss das Ergebnis mit dem aktuellsten mitgesendeten Hash übereinstimmen. Andernfalls muss die Datei neu gesendet werden.

3.4.3.5. File Delete Message

Eine File Delete Message beinhaltet den Pfad zur zu löschen Datei. Nach Empfang der File Delete Message wird die Datei gelöscht (siehe Abbildung 3.6, sblit – File Delete Message).

3.4.3.6. Device Refresh Message

Die Device Refresh Message dient zur Aktualisierung von Gerätelisten (siehe Abbildung 3.7, sblit – Device Refresh Message).

File-Flag

Das File-Flag gibt darüber Auskunft, ob es sich um die Datei für Partnergeräte oder die Datei für die eigenen Geräte handelt.

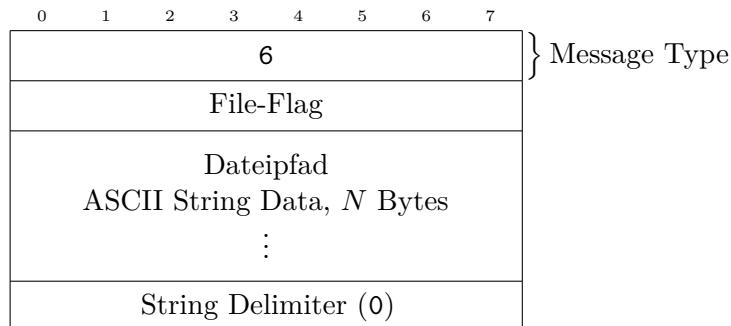


Abbildung 3.7.: sblit – Device Refresh Message

Geräte

Hier stehen die Adressen aller Geräte, die in der im File-Flag angegebenen Datei vorhanden sind. Bei Adressen von Partnergeräten werden nur jene mitgeschickt, die auch die eigenen Dateien speichern. Im Falle von eigenen Adressen, werden außerdem die Namen der Geräte, die man in der Konfiguration angegeben hat, mitgeschickt.

3.4.4. Nachrichten an Partnergeräte

- Partner File Requests
- Partner File Responses
- Partner File Messages
- Partner File Delete Messages
- File Delete Messages

3.4.4.1. Partner File Request

Der Partner File Request dient, wie der File Request, zum Versenden einer Dateianfrage (siehe Abbildung 3.8, sblit – Partner File Request Message).

Versionsverlauf

Der Versionsverlauf wird, wie beim File Request, mitgeschickt, damit der Partner die Datei identifizieren kann. Dies verhindert, dass der Partner den Dateipfad kennt und somit auf den Inhalt der Datei schließen kann. Weiters kann aus diesem Versionsverlauf, der lediglich Hashes enthält, nicht auf den Inhalt der Datei geschlossen werden, da das Verhashen eine Einwegfunktion ist.

3.4.4.2. Partner File Response

Der Partner File Response dient zum Antworten auf einen Partner File Request (siehe Abbildung 3.9, sblit – Partner File Response Message).

3. sblit

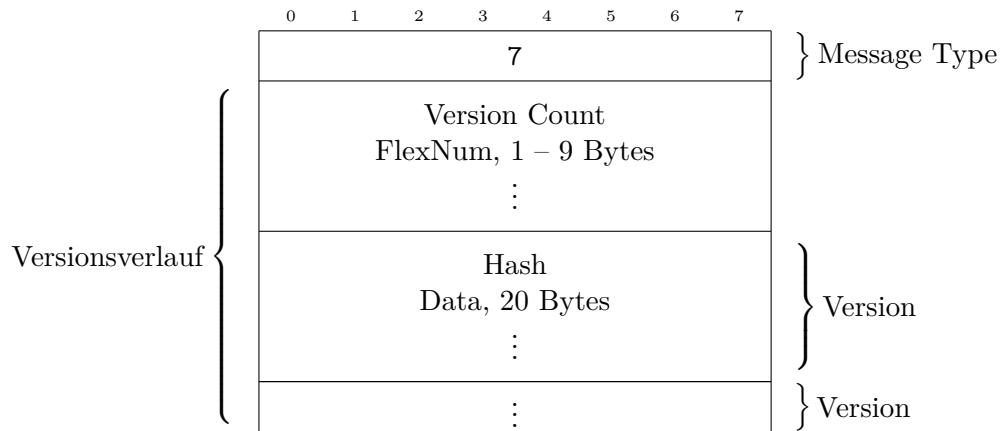


Abbildung 3.8.: sblit – Partner File Request Message

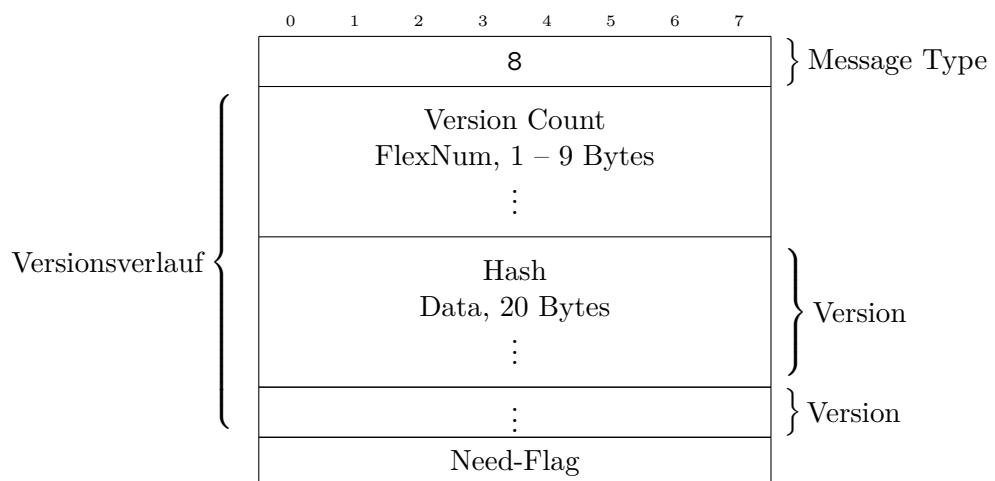


Abbildung 3.9.: sblit – Partner File Response Message

Versionsverlauf

Da die Datei auf den Partnergeräten mithilfe des Versionsverlaufs identifiziert wird, muss dieser mitgeschickt werden, um Verwechslungen vorzubeugen.

Need-Flag

Das Need-Flag im Partner File Response ist äquivalent zum Need-Flag im File Response (siehe 3.4.3.3 File Response, Seite 69).

3.4.4.3. Partner File Message

Die Partner File Message dient zum Versenden einer Datei an Partnergeräte (siehe 3.5.2 Partnergeräte, Seite 79) (siehe Abbildung 3.10, sblit – Partner File Message).

Versionsverlauf

Da der Versionsverlauf zur Identifikation der Datei benötigt wird, wird er auch bei der Partner File Message mitgeschickt. Gleichzeitig dient der letzte Hash im Versionsverlauf als Dateiname auf dem Partnergerät.

Dateinhalt

Um den Inhalt der Datei vor fremdem Zugriff zu schützen, wird er mit dem symmetrischen Schlüssel (siehe 3.5.5 Sicherheit, Seite 79) verschlüsselt.

Dateipfad

Der Dateipfad wird, wie auch beim Partner File Request, verschlüsselt übertragen.

Geräte mit der aktuellen Version

Hier stehen die Adressen aller Geräte, die schon die neueste Version haben. Ist die Version auf allen Geräten aktuell, kann sie von den Partnergeräten gelöscht werden. Daher wird diese Liste an Geräten immer mit der Datei mitgeschickt. Außerdem können somit unnötige Anfragen an Geräte, die die Datei schon besitzen, verhindert werden.

3.4.4.4. Partner File Delete Message

Dateipfad

Das einzige Attribut der Partner File Delete Message ist der verschlüsselte Dateipfad. Dieser wird eine Woche lang gespeichert und dann gelöscht.

Die Partner File Delete Message dient, wie die File Delete Message, zum Löschen von Dateien auf den eigenen Geräten (siehe Abbildung 3.11, sblit – Partner File Delete Message).

3.4.4.5. File Delete Partner Message

Um Platz auf den Partnergeräten zu sparen, werden die Dateien gelöscht, sobald diese auf allen eigenen Geräten verteilt sind. Dies wird mit der File Delete Message initiiert. In der File Delete Message zwischen Partnergerät und eigenem Gerät wird der Versionsverlauf statt dem Dateinamen angegeben (siehe Abbildung 3.6, sblit – File Delete Message).

Versionsverlauf

Der Versionsverlauf gibt die Datei an, die vom Partnergerät gelöscht werden soll.

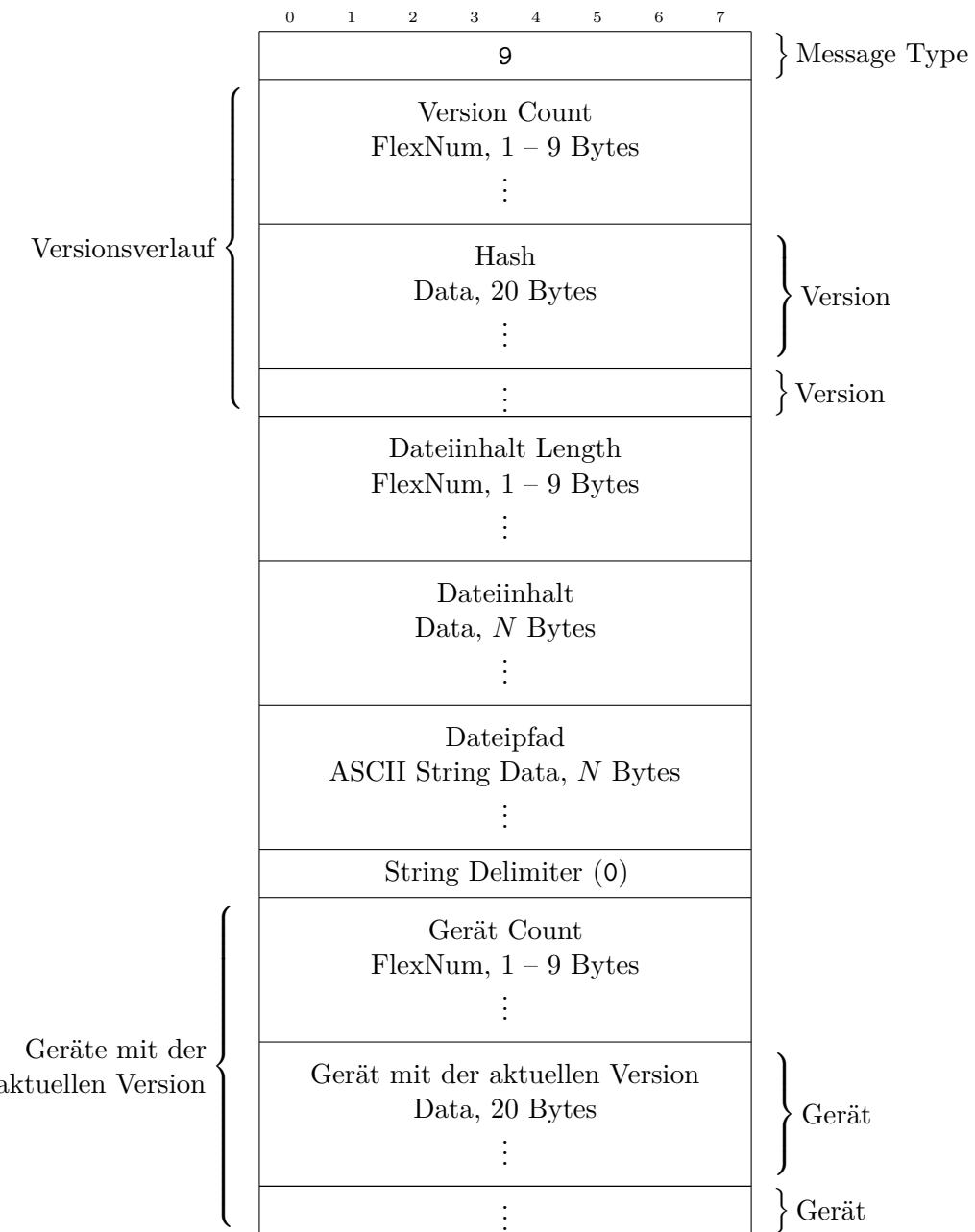


Abbildung 3.10.: sblit – Partner File Message

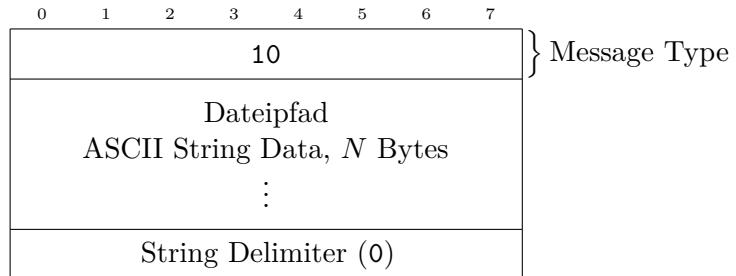


Abbildung 3.11.: sblit – Partner File Delete Message

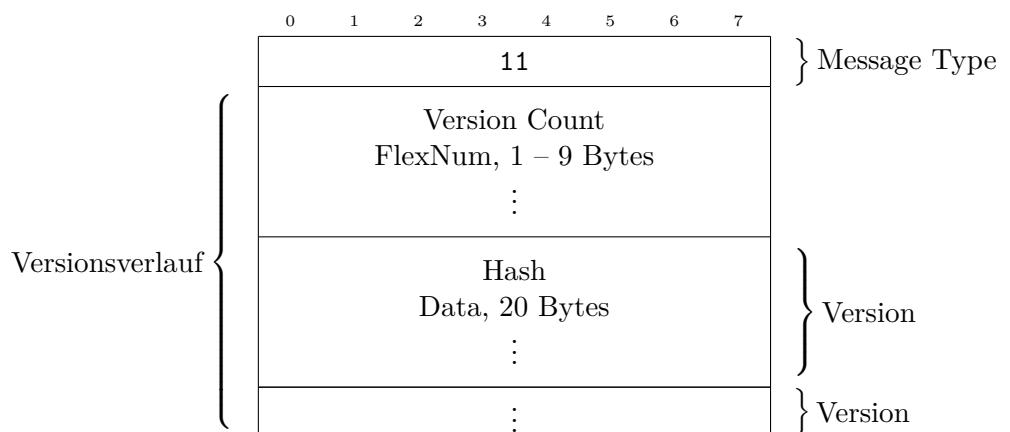


Abbildung 3.12.: sblit – File Delete Partner Message

3.4.5. Ablauf

3.4.5.1. Allgemein

Bevor eine Verbindung nicht autorisiert ist, werden sämtliche Nachrichten verworfen. Die einzige Außnahme hierbei sind Nachrichten, die zur Authentifizierung dienen.

3.4.5.2. Authentifizierung

Beim Programmstart versucht sich sblit zu allen eigenen Geräten und Partnergeräten zu verbinden. Dazu schickt es über DCL Application Channel Anfragen und fügt die Adresse des anderen Gerätes einer Liste mit unautorisierten Application Channels hinzu. Wird die Anfrage angenommen, wird sofort ein Authenticity Request verschickt.

Wird der Application Channel von einem anderen Gerät angefragt, wird zunächst in der Liste der eigenen Geräte (siehe 3.2.4 receivers.txt, Seite 58) und der Liste der Partnergeräte (siehe 3.2.2 freceivers.txt, Seite 55) nachgeschaut, ob die Adresse des Gerätes, das die Anfrage verschickt hat, in dieser vorhanden ist. Ist sie nicht vorhanden, wird der Application Channel sofort verworfen. Andernfalls wird ein Authenticity Request über den Application Channel verschickt und die Adresse des Gerätes, das den Application Channel angefragt hat, zur Liste an unautorisierten Application Channels hinzugefügt.

Wird ein Authenticity Request empfangen, werden die angefragten 64 Byte durch zufällige weitere 64 Bytes ergänzt. Anschließend werden die insgesamt 128 Bytes mit dem eigenen Private-Key verschlüsselt. Schließlich antwortet sblit mithilfe des Authenticity Responses auf den Authenticity Request.

Beim Empfangen des Authenticity Responses entschlüsselt sblit die Daten mithilfe des Public-Keys des Kommunikationspartners. Von den erhaltenen Daten werden die ersten 64 Bytes mit den ursprünglich gesendeten 64 Bytes verglichen. Stimmen die beiden Werte überein, konnte damit das Gerät seine Authentizität beweisen und die Adresse des Kommunikationspartners wird der Liste der authentifizierten Geräte hinzugefügt. Stimmen diese jedoch nicht überein, handelt es sich um einen Betrüger, der offensichtlich den richtigen Private-Key zur von ihm angegebenen Adresse nicht kennt.

3.4.5.3. Änderung einer Datei

Bevor die eigentliche Datei übertragen werden kann, muss zunächst ein File Request an den Kommunikationspartner gesendet werden. Das Senden des File Requests wird entweder direkt vom WatchService oder beim Autorisieren eines Application Channel initiiert.

Nach Empfang des File Request wird zunächst geprüft, ob die Datei vorhanden und aktuell ist. Dazu wird der Versionsverlauf im File Request mit dem Versionsverlauf im lokalen Logfile verglichen. Ist die aktuellste Version der Datei im mitgeschickten Versionsverlauf nicht im lokalen Versionsverlauf enthalten, wird das Need-Flag auf den Hexadezimalwert 0x01 gesetzt. Stimmen die aktuellsten Hashes in empfangenem und lokalem Versionsverlauf überein, wird es auf den Hexadezimalwert 0x00 gesetzt und die Adresse des Partners im Logfile der Liste der Geräte mit der aktuellsten Version hinzugefügt. Des Weiteren wird überprüft, ob ein Konflikt aufgetreten ist (siehe 3.3.5.2 Konflikterkennung, Seite 64). Anschließend wird der File Response verschickt.

Nach Empfang des File Response wird zunächst überprüft, ob der Hashwert der Datei im Logfile mit dem erhaltenen Pfad übereinstimmt. Stimmt der Hashwert im File Response

nicht mit dem aktuellen lokalen Hashwert überein, wird die Antwort verworfen. Dies kann beispielsweise passieren, wenn in der Zwischenzeit eine neuere Version der Datei erzeugt wurde. Stimmt dieser jedoch überein, kann die Datei nun im nächsten Schritt mithilfe der File Message verschickt werden.

Wird ein File Response empfangen, wird zunächst das Logfile bearbeitet. Dazu wird der ursprüngliche Eintrag zur empfangenen Datei samt dem Versionsverlauf und den Geräten mit der aktuellen Version durch die neuen Parameter ersetzt. Dies passiert, damit die Änderung vom WatchService nicht fälschlicherweise als Änderung durch den Benutzer wahrgenommen wird. Anschließend wird der empfangene Dateiinhalt vorerst in eine temporäre Datei geschrieben. Diese versteckte, temporäre Datei wird mit dem Prefix *.sblit* gespeichert, damit bei einem Absturz keine Dateien verloren gehen. Nach dem Schreiben der Datei wird sie in den ursprünglichen Namen umbenannt.

3.4.5.4. Löschen einer Datei

Das Senden einer File Delete Message wird, wie das Senden eines File Request, entweder durch das WatchService oder beim Autorisieren eines Application Channel initiiert.

Beim Empfangen dieser Anfrage wird der Dateiname mit einem Zeitstempel in einer temporären Datei gespeichert, damit die Datei auch auf Geräten, die gerade nicht online sind, gelöscht werden kann. Außerdem wird die angefragte Datei gelöscht, sobald die Anfrage empfangen wird. Deshalb sollte man eine alte Version nicht löschen, nur weil sie nicht auf dem aktuellsten Stand ist, denn dies löscht ebenfalls die Datei auf dem aktuellsten Stand.

3.4.5.5. Aktualisieren einer Gerätedatei

Die Gerätedateien werden ebenfalls von einem Watchservice überwacht. Werden diese bearbeitet, wird eine Device Refresh Message an die anderen Geräte, die gerade online sind, verschickt. Außerdem wird diese periodisch in einem konfigurierbaren Intervall den anderen Geräten mitgeteilt. Beim Empfangen wird überprüft, ob die empfangene Version aktueller ist. Im Falle, dass die Datei aktueller ist, werden die Neuerungen in die Datei geschrieben.

3.4.5.6. Speichern einer Datei auf einem Partnergerät

Wird eine Datei von einem Benutzer verändert, wird diese, sofern nicht alle eigenen Geräte online sind, auf ein Partnergerät übertragen. Dazu wird ein Partner File Request an das Partnergerät geschickt.

Beim Empfangen des Partner File Request wird überprüft, ob eine Datei mit dem in der Anfrage mitgeschickten Versionsverlauf noch nicht existiert und ob genug von dem zur Verfügung gestellten Speicherplatz vorhanden ist. Werden beide Bedingungen erfüllt, wird das Need-Flag auf den Hexadezimalwert 0x01 gesetzt und gemeinsam mit dem Versionsverlauf in einem Partner File Response zurückgeschickt. Andernfalls wird mit dem Versionsverlauf und dem Hexadezimalwert 0x00 geantwortet.

Nachdem der Partner File Response empfangen wurde, wird, falls das Need-Flag auf 0x01 gesetzt ist, mit der Partner File Message geantwortet. Bevor die Partner File Message verschickt wird, werden der Dateiinhalt, Dateipfad und die Geräte mit der aktuellen Version mit dem symmetrischen Schlüssel verschlüsselt (siehe 3.5.5 Sicherheit, Seite 79).

Beim Empfangen des Partner File Request wird der aktuellste Hash gekürzt. Anschließend wird der verschlüsselte Dateiinhalt in einer Datei mit dem Namen des gekürzten Hashes gespeichert. Anschließend werden Dateipfad und die Geräteliste ebenfalls verschlüsselt gespeichert.

3.5. Partnerschaften

3.5.1. Allgemein

Um nicht immer mindestens zwei Geräte eingeschalten haben zu müssen, werden die Daten nicht nur auf den eigenen Geräten gespeichert, sondern auch auf sogenannten Partnergeräten. Damit man trotz hoher Verfügbarkeit nicht zu viel Speicher freigeben muss, sollte man Partnerschaften mit circa 10 Geräten eingehen.

3.5.2. Partnergeräte

Ein Partnergerät zeichnet sich dadurch aus, dass entweder meine Daten auf dem Partnergerät oder dessen Daten am eigenen Gerät zwischengespeichert werden. Zwischen den Partnergeräten bestehen sogenannte Partnerschaften. Diese werden zwischen zwei Benutzern geschlossen und erlauben es dem jeweils anderen, Dateien auf einem eigenen Gerät zu speichern.

3.5.3. Delta-Daten

Um den auf den Partnergeräten benötigten und somit auch den freigegebenen Speicherplatz möglichst gering zu halten, synchronisiert sblit nur Delta-Daten. Unter Delta-Daten werden die Änderungen an einer Datei verstanden. Bis zur Umsetzung des Erkennungsalgorithmus für Delta-Daten werden nur die veränderten Dateien synchronisiert.

Wenn sich eine Datei ändert wird diese so lange auf den Partnergeräten gespeichert, bis diese auf allen Geräten verfügbar ist.

3.5.4. Dauer einer Partnerschaft

Da manche Geräte von Zeit zu Zeit mehr oder weniger genutzt werden, können Partnerschaften bei Inaktivität eines Nutzers gekündigt werden. Dazu löscht man einfach das Partnergerät aus der Liste. Beim Löschen kommt das Partnergerät auf eine Blacklist. Bei der nächsten Anfrage des Partnergeräts wird die Verbindung nicht mehr zugelassen und das Gerät merkt, dass die Partnerschaft nicht mehr gewünscht ist. Danach wird die Partnerschaft auch vom Partnergerät gelöscht.

3.5.5. Sicherheit

Um zu verhindern, dass eine eigene Datei vom Benutzer eines Partnergerätes gelesen wird, wird diese mit dem AES im GCM verschlüsselt. Des Weiteren wird dadurch verhindert, dass die Datei durch den Partner verändert werden kann.

3.5.6. Wunschpartnerschaften

Kennen sich zwei sblit-User und haben einen ähnlichen Verwendungsrythmus, können diese eine Partnerschaft miteinander schließen. Dabei müssen die beiden nur ihre Adressen austauschen und zur Liste der Partnergeräte hinzufügen. Im weiteren Verlauf verhalten sich Wunschpartnerschaften nicht anders als andere Partnerschaften. Diese können, wie auch andere Partnerschaften, bei Bedarf gelöscht werden.

3.5.7. Beispiel

Susanne und Wilfried benötigen jeweils ein Gigabyte Synchronisationsspeicher. Susanne wählt eines ihrer Geräte aus, auf dem sie mehr als ein Gigabyte Speicher frei und genug Bandbreite zur Verfügung hat (im Folgenden Gerät A(S)). Wilfried sucht sich ebenfalls ein Gerät aus, das die Bedingungen erfüllt (im Folgenden Gerät A(W)). Nun tauschen die

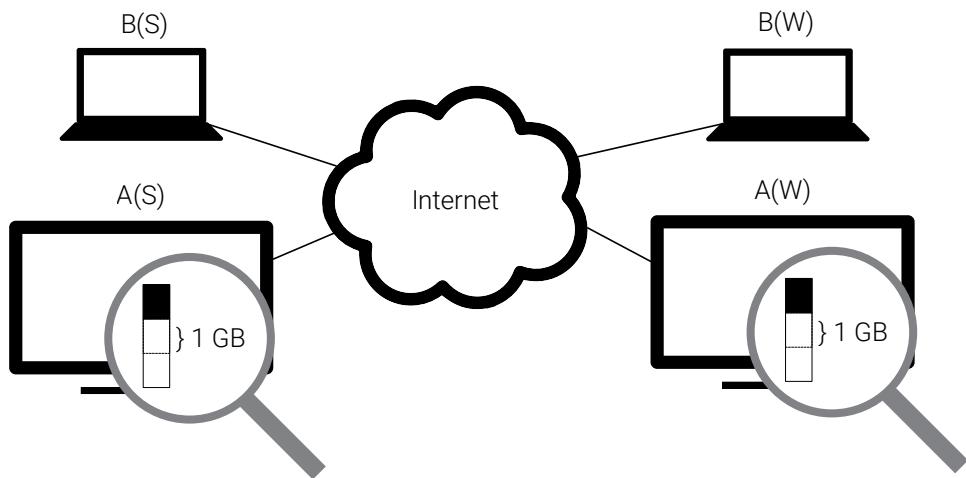


Abbildung 3.13.: Wahl eines Geräts, auf dem Speicherplatz freigegeben wird.

Geräte A(S) und A(W) ihre Adressen aus und tragen sie beim jeweils anderen in die Liste der Partnergeräte ein. Sobald beide Geräte gleichzeitig online sind, werden alle Geräte von Susanne auf Gerät A(W) und umgekehrt alle Geräte von Wilfried auf Gerät A(S) gespeichert. Weiters wird die Adresse von A(W) auf Susannes Geräte und die Adresse von A(S) auf Wilfrieds Geräte gespeichert. Ab diesem Zeitpunkt werden alle Änderungen von Susannes Dateien auf Gerät A(W) gesichert, bis sie auf alle Geräte von Susanne verteilt wurden. Sobald Susannes Geräte die Änderung erhalten, wird Gerät A(W) wieder aufgefordert, die Daten zu löschen. Das gleiche gilt natürlich auch für Wilfried und das Gerät A(S).

3. sblit

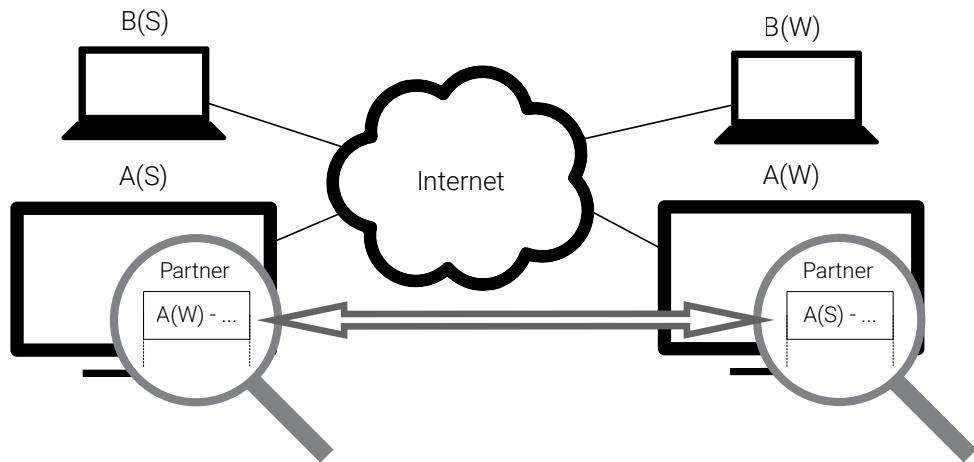


Abbildung 3.14.: Austausch und Eintragen der Adressen.

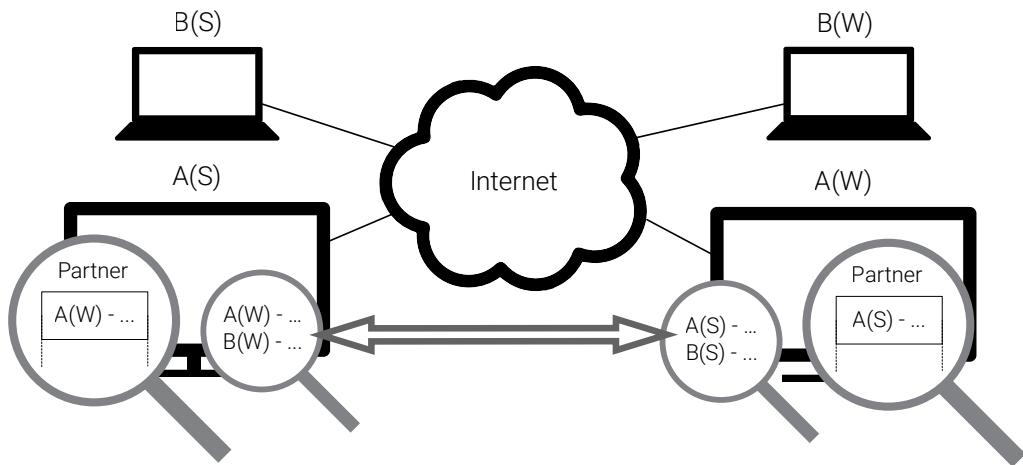


Abbildung 3.15.: Eintragen der Geräte des Partners.

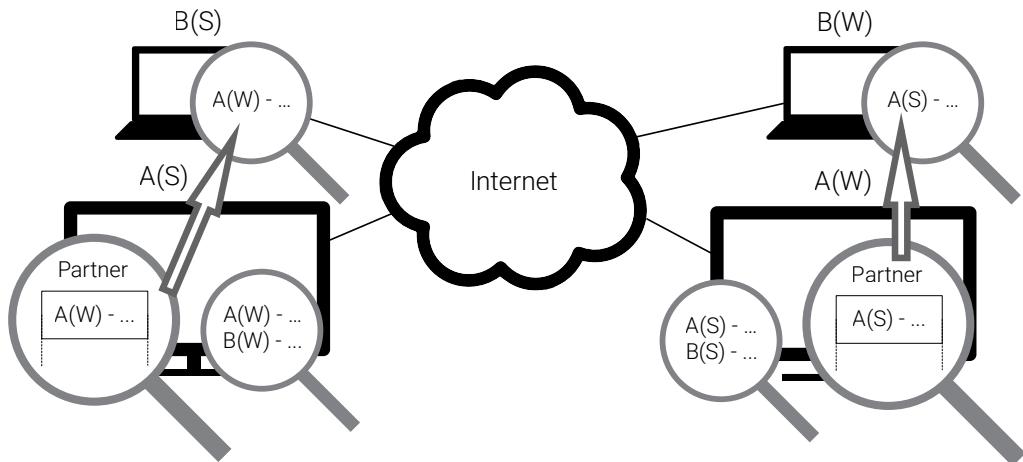


Abbildung 3.16.: Eintragen der Adressen des Partnergeräts.

4. Graphical User Interface

4.1. Einleitung

Mit dem Gedanken, dass ein Dateisynchronisationstool hauptsächlich im Hintergrund arbeitet, ist die grafische Benutzeroberfläche zurückhaltend und einfach gestaltet. Dem Nutzer werden nicht unnötig viel Konfigurationsmöglichkeiten gegeben, damit er leicht den Überblick behalten kann. Beim Starten von sblit erscheint das Icon im System-Tray, welches als Angelpunkt für jegliche Nutzerinteraktion dient.

4.2. System-Tray

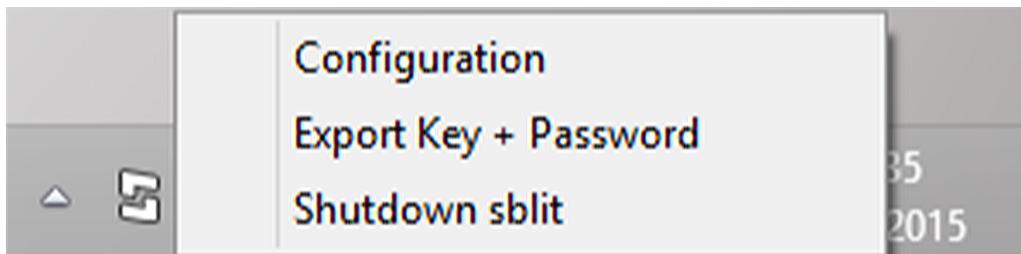


Abbildung 4.1.: Menüfenster beim Systemtray

Mit einem Rechtsklick auf das Icon im System-Tray öffnet sich eine kleine Menüauswahl, bei der folgende Punkte zur Auswahl stehen:

Configuration Öffnet das Konfigurationsmenü.

Export Key Öffnet einen Dialog zur Auswahl des Speicherorts. In der exportieren Text-Datei befindet sich der öffentliche Schlüssel des eigenen Gerätes und der symmetrische Schlüssel für alle Synchronisationspartner.

Shutdown sblit Stoppt alle laufenden Synchronisationen und beendet die Anwendung.

4.3. Überblicksfenster

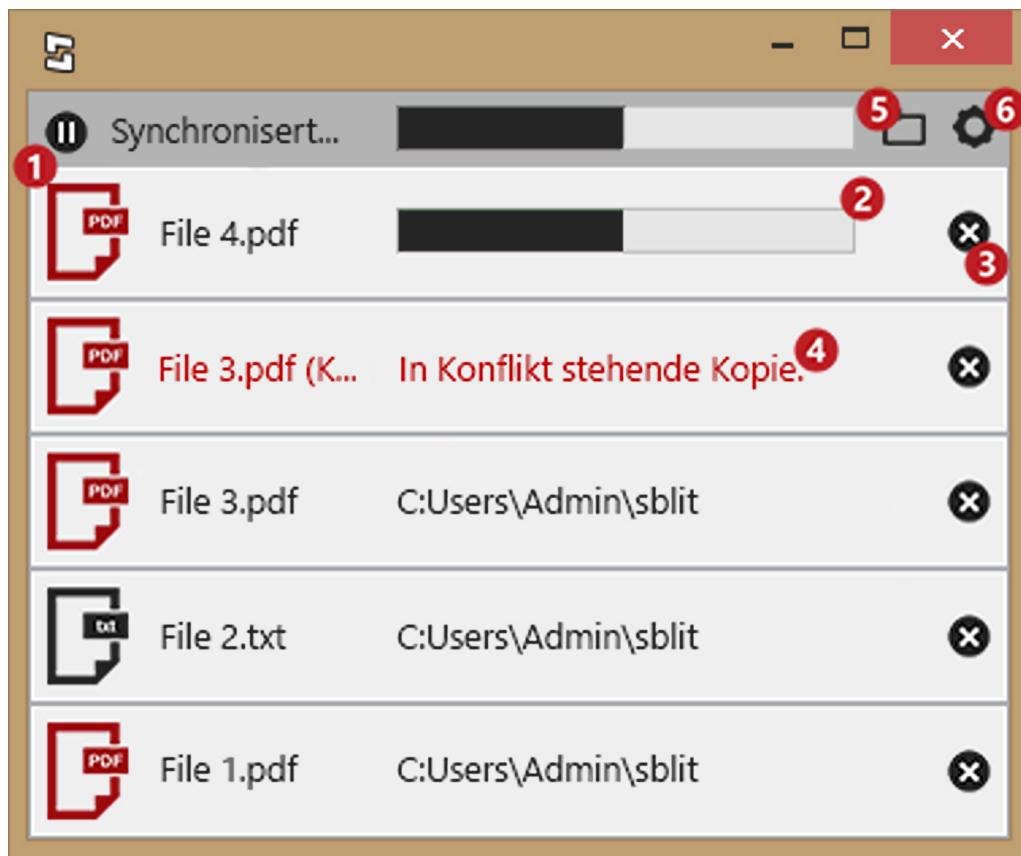


Abbildung 4.2.: Überblicksfenster

Mit einem einfachen Klick auf das Icon im System-Tray öffnet sich das Überblicksfenster, in der der Benutzer auf die folgenden Optionen Zugriff bekommt:

Letzte Änderungen innerhalb des sblit-Ordners Dem Benutzer wird hier eine Auf-
listung der zuletzt hinzugefügten Dateien geboten. Neben dem an den Dateityp
angepassten Bild wird auch der Dateiname und Ordnerpfad angegeben.

Fortschrittsbalken laufender Synchronisationsvorgänge Bei laufenden Synchroni-
sationen hat der User die Möglichkeit, den Fortschritt zu verfolgen.

Button für das Abbrechen der Synchronisation Beispielsweise nach irrtümlichem
Hinzufügen von Dateien, hat der Benutzer die Möglichkeit, die laufende Synchronisa-
tion mithilfe des Abbrechen-Buttons abzubrechen.

Anzeige von aufgetretenen Fehlern Bei Synchronisationskonflikten, die auftreten,
wenn zwei Synchronisationspartner die selbe Datei gleichzeitig bearbeiten, sowie
bei diversen anderen Fehlern, wird der User benachrichtigt.

4. Graphical User Interface

Link zum sblit-Ordner Um dem Benutzer schnellen Zugriff auf seinen konfigurierten sblit-Ordner zu gestatten, gibt es den Ordner-Button, mit dem sich der sblit-Ordner im Datei-Browser öffnet.

Öffnen des Konfigurationsmenüs Mit einem Klick auf den Optionen-Button öffnet sich das Konfigurationsmenü, welches im folgenden Abschnitt gezeigt wird.

4.4. Konfigurationsmenü

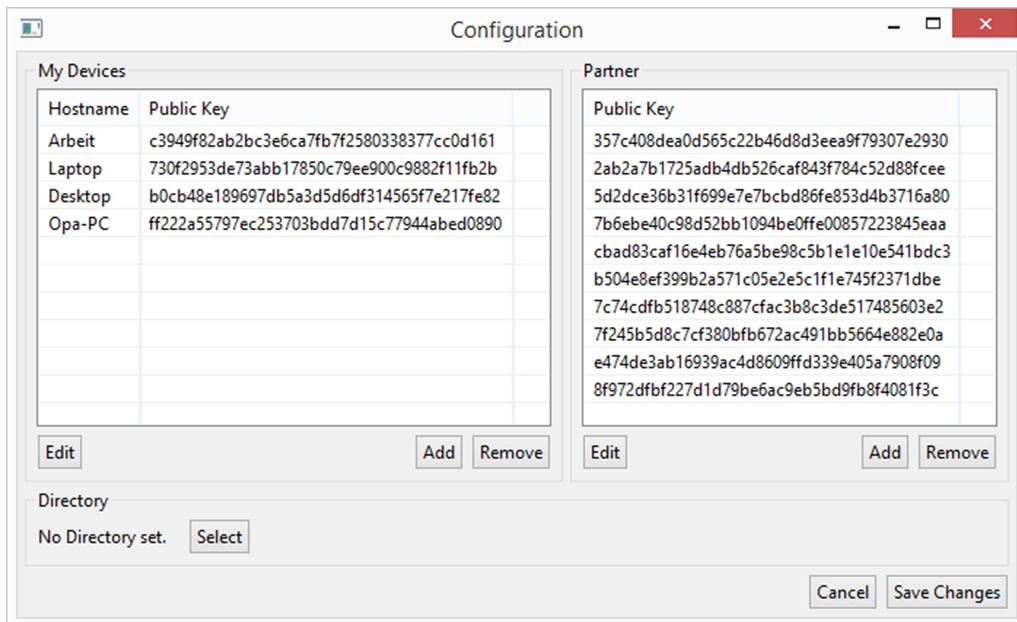


Abbildung 4.3.: Konfigurationsmenü

Geräte zu den Synchronisationspartnern hinzufügen/entfernen Unter dem Punkt “My Devices” befindet sich die Liste der Synchronisationsgeräte, also jene Geräte, mit denen der unter Punkt 3 angegebene sblit-Ordner synchronisiert wird. Die Einträge bestehen aus dem vom Benutzer angegebenen Namen des Geräts und dessen Public-Key, welcher seine Adresse darstellt. Der Benutzer hat die Möglichkeit, bestehende Einträge für Korrekturen zu bearbeiten, neue hinzuzufügen oder alte zu entfernen.

Manuelles Eintragen von Partnergeräten Obwohl die Liste der Partnergeräte automatisch von sblit verwaltet wird, hat der Benutzer die Möglichkeit, Geräte manuell hinzuzufügen. Während nämlich normalerweise Geräte fremder Nutzer als Partnergeräte fungieren, kann man sich auch gegenseitig mit einer Freundin oder einem Freund Speicherplatz freigeben, indem man den Public-Key des jeweils anderen in der Liste der Partnergeräte einträgt.

Verschieben des sblit-Ordners Der zu synchronisierende Ordner kann im Konfigurationsmenü auch nach der Installation noch geändert und verschoben werden.

4.5. Konfigurationsmenü

4.5.1. Einführung

Für das gestalten der grafischen Oberfläche wurde bei sblit SWT verwendet. Das Standard Widget Toolkit ist eine Open-Source-Bibliothek für grafische Benutzeroberflächen entwickelt von Eclipse. Es unterstützt viele Plattformen, darunter Windows 2000/XP/Vista/7/8, Linux und Mac OS X und bietet eine Programmschnittstelle, die es erlaubt, auf die nativen Widgets des Betriebssystems zuzugreifen, sofern dies möglich ist. Zusätzlich werden noch weitere Widgets angeboten, die nicht nativ existieren, aber in dem Betriebssystem-spezifischen Design gehalten sind, wie zum Beispiel das Tree- oder Table-Widget.

4.5.2. Vorteile

Behält Design der jeweiligen Plattform Damit eine möglichst große Reichweite mit sblit erreicht werden kann, wurde bei besonders auf Plattformunabhängigkeit geachtet, um sblit mit wenig Anpassungsaufwand auf verschiedenen Betriebssystemen laufen lassen zu können. SWT hat nun den Vorteil, dass es auf den verschiedenen Plattformen das spezifische Aussehen des jeweiligen Betriebssystems hat, obwohl das selbe Programm ausgeführt und die selben Elemente verwendet werden.

Einstiegerfreundlicher Editor Sich mit einem neuen Framework zu beschäftigen kostet Zeit und Energie. Um diesen Schritt zu erleichtern, gibt es für die meisten Frameworks für grafische Oberflächen einen What-you-see-is-what-you-get-Editor. So auch bei SWT, weshalb sich die Einarbeitungsphase einfacher gestaltet, als dies bei manch anderen Frameworks der Fall ist.

4.5.3. Nachteile

Ist auf Linux weniger performant Auf allen Plattformen außer Windows (wie zum Beispiel Linux) ist SWT manchmal weniger performant, da bestimmte Widgets nicht vorhanden sind und erst emuliert werden müssen.

Ist nicht MVC basiert SWT basiert nicht auf einem MVC-Design, bietet mit JFace allerdings eine MVC-Abstraktion als Erweiterung.

4.5.4. Aufbau

Der Aufbau bei SWT ist sehr einheitlich. Es gibt zwei Hauptelemente: `Display` und `Shell`. Jedes Element hat ein Partnerelement als Parameter und jedes Widget noch dazu ein Stil-Bit, dass dem Element eine vordefinierte Optik verleiht. Zudem gibt es noch Container-Klassen, auf die ein Layout angewendet werden kann, um die ihm untergeordneten Elemente zu positionieren.

```

1 public class erklaerungsGUI {
2     Display display;
3     Shell shell;
4
5     public static void main(String[] args) {
6         new erklaerungsGUI();

```

```

7     }
8
9     public erklaerungsGUI(){
10    display = new Display();
11    shell = new Shell(display);
12
13    GridLayout layout = new GridLayout(2, false);
14    shell.setLayout(layout);
15
16    Button pushButton = new Button(shell, SWT.PUSH);
17    GridData layoutData = new GridData(SWT.FILL, SWT.FILL
18        , true, true, 1, 1);
19    pushButton.setLayoutData(layoutData);
20    pushButton.setText("Push");
21
22    Button arrowButton = new Button(shell, SWT.ARROW);
23    layoutData = new GridData(SWT.FILL, SWT.FILL, true,
24        true, 1, 1);
25    arrowButton.setLayoutData(layoutData);
26
27    shell.setSize(200, 100);
28    shell.open();
29
30    // Eventloop
31    while (!shell.isDisposed()) {
32        if (!display.readAndDispatch())
33            display.sleep();
34    }
35}

```

Listing 4.1: Beispielklasse für grafische Oberfläche *erklaerungsGUI*

Display Die **Display**-Klasse ist einer der Kernkomponenten innerhalb einer SWT-GUI.

Sie verwaltet Event-Loops, Schriftarten, Farben und die generelle Kommunikation zwischen dem GUI-Thread und den anderen Threads. Jedes Programm mit einer SWT-GUI muss ein **Display**-Objekt haben.

Shell Die **Shell**-Klasse ist die zweite der Kernkomponenten einer SWT-GUI. Sie stellt das Programmfenster dar und wird das Elternelement aller eingefügten Widgets. Das **Display**-Objekt muss ein oder mehrere **Shell**-Objekte als Unterobjekt haben. Mit der Funktion **shell.open()** wird das Programmfenster erst gezeichnet.

GridLayout Layouts können auf **Composite**-Objekte angewandt werden, um die darin befindlichen Elemente zu positionieren. Ein Beispiel für ein solches Objekt ist die **Shell**. Es gibt verschiedenste Layouttypen, je nach Struktur, in der man die Elemente

anordnen will. Um jedem Widget eine Layoutinformation, also unter anderem eine Position zuordnen zu können, die jeweili spezielle `LayoutData` benötigt.

Button Zwei typische Buttons als Beispiel-Widgets. Als Parameter wird die Shell als Elternelement angegeben und `SWT.PUSH` beziehungsweise `SWT.ARROW` als Stil-Bit. Dieser unterschiedliche Parameter erklärt auch die andere Optik der Buttons.

Eventloop Damit das GUI nicht sofort geschlossen wird, gibt es diese Eventloop. Eine Schleife, die wartet bis ein gewisses Event eintritt. In diesem Fall wird darauf gewartet, dass das Fenster, also das `Shell`-Objekt disposed, also geschlossen wird.

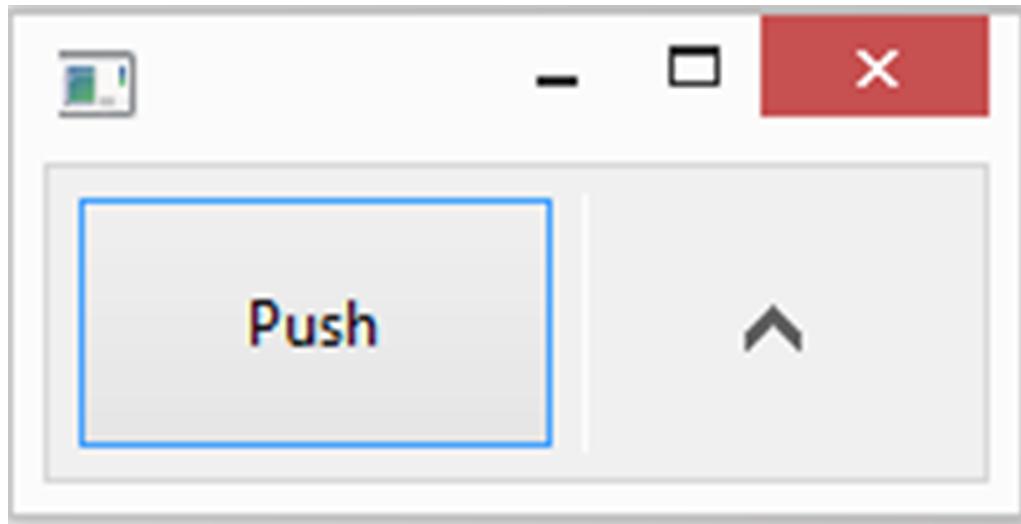


Abbildung 4.4.: Beispiel-GUI zur Erklärung

5. Website

5.1. Allgemein

Zu Beginn der Diplomarbeit galt es eine Website zu gestalten, auf der die Projektidee, die Teammitglieder und der Projektstatus einsehbar sind.

Primär lag aber die Applikation im Fokus der Diplomarbeit, weshalb Bootstrap, ein offenes CSS-Framework, verwendet wurde, das viele vordefinierte Auszeichnungen für HTML-Elemente aller Art bietet und damit den für estästische Websites zu erbringenden Arbeitsaufwand verringert.

Viele Frontend-Entwickler veröffentlichen ihre eigenen Bootstrap-Themen, also mit bootstrap-Klassen gestaltete statische HTML-Seiten und machen es Drittanwendern möglich, diese bestehenden Frontend-Webseiten für eigene Zwecke umzugestalten, um somit einen schönen und unkomplizierten Web-Auftritt zu ermöglichen. Namenhafte Webseiten, auf denen verschiedenste Bootstrap-Themen ausgestellt und unter anderem zur Verwendung frei gestellt sind wären startbootstrap.com, bootstrapzero.com oder blacktie.co.

5.2. Diplomarbeitswebsite

Der Webauftritt der Diplomarbeit ist aus solch einem Bootstrap-Thema von startbootstrap.com entstanden, welches optisch und inhaltsmäßig auf unsere Diplomarbeit angepasst worden ist. Dabei wurde ein, für Bootstrap typisches One-Page-Design verwendet.

Aufrufbar ist die Diplomarbeitswebsite unter sblit.net/da

5.3. Produktwebsite

Mit dem Ende der Diplomarbeit hat sblit nicht nur einen Webauftritt für das Projekt, sondern auch einen für das Produkt selbst, abseits der schulinternen Richtlinien für Diplomarbeitswebseiten.

Unter sblit.net/ ist die englischsprachige Produktwebsite aufrufbar, die Besucher der Seite über die wichtigsten Eckdaten unseres Produktes informiert und bei Abschluss des Projektes auch eine Download-Sektion haben wird, um sich einen Installationsclient herunterladen zu können.

Auch bei dieser Website wurde das, für Bootstrap typische One-Page-Design verwendet.

6. Wettbewerbe

6.1. Allgemein

Jedes Jahr gibt es für Studierende, Schülerinnen und Schüler die Möglichkeit, ihre privaten oder die ihm Rahmen ihrer Ausbildung entstandenen Projekte bei Projekt-Wettbewerben einzureichen. Für Diplomarbeiten wie sblit stellen diese Wettbewerbe eine einfache und effektive Methode dar um Marketing zu betreiben und den Bekanntheitsgrad des Projekts zu steigern.

Mit einem Projekt wie sblit, das eine kreative Idee besitzt, aktuelle Themen behandelt und technischen Tiefgang hat, finden sich leicht Interessenten unter den zahlreichen Besuchern von Projektwettbewerben.

Durch die Schule wurden die Mitglieder aller Diplomarbeiten über diverse Vertreter solcher Wettbewerbe informiert. Das sblit-Diplomarbeitsteam hat das Projekt daraufhin bei einigen Wettbewerben eingereicht. Der folgende Abschnitt behandelt die Wettbewerbe, bei denen das Projekt eingereicht und dafür erforderlichen Dokumente.

6.2. Wettbewerbe

6.2.1. Jugend Innovativ

6.2.1.1. Allgemein

Jugend Innovativ ist der größte Wettbewerb für innovative Schulprojekte in Österreich und bietet Jugendlichen die Möglichkeit, ihre innovativen Projekte von einer Fachjury bewerten zu lassen und mit etwas Glück unter den Gewinnern zu landen, denen ein Preisgeld von bis zu 2000€ zugesprochen wird.

6.2.1.2. Einreichung

Nachdem sblit in der Kategorie idea.goes.app eingereicht wurde musste für den Wettbewerb ein Projektstatusbericht verfasst werden, der einen einen genauen Einblick in das Projekt geben soll und zusätzlich auch die Ergebnisse aufzeigt, die bis dato erzielt wurden.

6.2.1.3. Ergebnis

Vorerst (Stand April 2015) sind nur die Halbfinalisten der teilnehmenden Projekten bekannt. Aus diesem Grund wurde das Diplomarbeitsteam zum gleichnamigen Halbfinal-Event eingeladen, um das Projekt vorort auszustellen.

Um den eigenen Projektstand zu schmücken wurde ein Plakat angefertigt, auf dem die Eckdaten unseres Projekts einsehbar sind. Zusätzlich wurden zum erleichterten Weiterreichen der Kontaktinformationen des Projektteams Visitenkarten entworfen und gedruckt (siehe C Anhang 3, Seite 123).

6.2.2. u19 – Create your World

6.2.2.1. Allgemein

u19 – CREATE YOUR WORLD ist der Jugendwettbewerb des Prix Ars Electronica. Hier wird die Möglichkeit geboten, eigene Vorstellungen und Ideen zu realisieren und zu präsentieren.

6.2.2.2. Einreichung

Ursprünglich wurde das Projekt selbst als Dateiabgabe oder eine technische Dokumentation als Abgabe gefordert. Da sblit zu diesem Zeitpunkt aber noch nicht fertiggestellt wurde und auch nicht als einzelne Datei abgegeben hätte werden können, einigte man sich mit den Organisatoren darauf, einen Projektfolder einzureichen, der die Projektidee beschreibt und noch nähere technische Details zu den einzelnen Teilgebieten und den Projektstatus beinhaltet (siehe B Anhang 2, Seite 101).

6.2.2.3. Ergebnis

Die Ergebnisse wurden noch nicht bekanntgegeben (Stand April 2015).

6.2.3. AXAWARD

6.2.3.1. Allgemein

AXAWARD steht für AUSTRIAN X.TEST AWARD und ist ein Wettbewerb für technische Schulprojekte und Forschungsarbeiten von Teams mit bis zu 3 Mitgliedern und wird von der österreichischen Messtechnik-Firma x.test veranstaltet.

6.2.3.2. Einreichung

Auch beim AXAWARD wurde das Projekt selbst als Dateiabgabe oder eine technische Dokumentation als Abgabe gefordert. Aufgrund dessen wurde auch hier der Projektfolder eingereicht (siehe B Anhang 2, Seite 101).

6.2.3.3. Ergebnis

Die Finalteams des AXAWARD 2015 wurden im April 2015 per Mail bekanntgegeben, wobei es sblit nicht unter die besten zehn Projekte geschafft hat. Genaue Platzierungen wurden nicht bekannt gegeben.

6.2.4. Internet of Things Cup

6.2.4.1. Allgemein

Der Internet of Things Cup ist ein Open Source Projektwettbewerb für Studierende, Schülerinnen und Schüler, der den Fokus auf das namensgebende Internet der Dinge legt. sblit eignet sich insofern für den Wettbewerb, als dass, basierend auf einzelnen Komponenten der Umsetzung, dezentrale Speicher- und Kommunikationsmöglichkeiten geschaffen werden können, die in Internet of Things und Machine-to-Machine-Anwendungen integriert werden können.

6. Wettbewerbe

Für das Projekt wurde dem Diplomarbeitsteam kostenlose Hard- und Software zur Verfügung gestellt: ein Beaglebone-Mikrocomputers, eine Jahreslizenz für Microsofts Azure Datenbank und Conrad Gutscheine.

6.2.4.2. Einreichung

Im April war die Abgabe des Projekt-Zwischenberichts, welcher die erste Abgabe nach der Einreichung darstellt. Der Bericht setzt sich aus einem Google-Drive-Fragebogen zusammen, bei dem grundsätzliche Informationen an die Wettbewerbsleitung übergeben werden, wie zum Beispiel der Projektstatus in Prozentangabe oder auch das Beilegen von Grafiken des Projekts. Zudem waren noch relevante Links wie zum Beispiel zu der Projekt-Website oder Social-Media-Seiten anzugeben.

6.2.4.3. Ergebnis

Ergebnisse wurden noch nicht bekanntgegeben, da dieser Wettbewerb noch am Laufen ist. Die Siegerehrung des IoT Cups findet am 3. Juli 2015 statt.

6.2.5. Computer Creative Wettbewerb

6.2.5.1. Allgemein

Beim Computer Creative Wettbewerb, der von der Österreichischen Computer Gesellschaft organisiert wird, ist die Ausarbeitung eines Projekts, das sich kreativ mit Informatik auseinandersetzt das Ziel. Die Bewerber dürfen dabei nur jünger als 21 Jahre sein.

Die eingereichten Projekte kommen aus verschiedensten Teilbereichen der Informatik, sodass sich diverse Arbeiten mit Multimedia, Internet, Robotik, Webseitengestaltung, Spielen oder Programmieren als Thematik gegenüberstehen und miteinander verglichen werden.

Für die ersten fünf in jeder Kategorie gibt es Geldpreise zu gewinnen.

6.2.5.2. Einreichung

sbliit zählt zur Sekundarstufe II, also Oberstufe von 14 bis 20 Jahren. Zur Teilnahme musste ein Online-Formular ausgefüllt und eine detaillierte Projektbeschreibung verfasst und abgegeben werden. Auch bei diesem Wettbewerb wurde der Projektflyer abgegeben (siehe B Anhang 2, Seite 101).

6.2.5.3. Ergebnis

Einsendeschluss der Projekte war bis April, weshalb dieser Wettbewerb noch einige Zeit andauern wird. Genaue Termine sind noch nicht bekannt.

A. Anhang 1

DCL Integration Tutorial

Martin Exner

April 21, 2015

This article aims at providing a simple tutorial on how to use the Decentralized Communication Layer with third party applications written in Java.

Introduction

The Decentralized Communication Layer (DCL) is a network of decentralized peer-to-peer networks that can be used to route communication of third party applications through secure and private channels. Each of these networks has a unique network identifier.

At the time of writing, the only network defined for DCL is the circle network with the identifier `org.dclayer.circle`. In this network, each node has an address which is computed by hashing the RSA public key of an RSA keypair the node generates at startup. SHA-1 is used as hash algorithm, which yields addresses that are 20 bytes in length. Those addresses can then be validated by performing a crypto challenge with the node that should be checked.

Messages in the circle network are routed in a way similar to the Kademlia model. Each node forwards the message to the neighbor with the address that is numerically closest to the message's destination address. In order for this to work, each node needs to be connected to many nodes with addresses that are numerically close to its own address and each node must be con-

nected to the two nodes that have addresses with the shortest possible distance to its own. The amount of connections to nodes with numerically distant addresses does affect the number of hops required for routing messages, but does not influence routing as much.

Communication

There are two ways application instances can communicate using the DCL. The first is via unreliably transmitted packets, which are routed through a specific DCL network and may or may not arrive at their destination. The second method is via encrypted and connected application channels, which provide reliable transmission of data. For both the initiation of application channels and the transmission of unreliable packets over the circle network, the public key of the remote node is required as destination address.

Overview

In order to integrate DCL communication features in third party applications, DCL provides a Java library that manages the TCP connection to the DCL service, including creating and accepting application channels and sending and receiving unreliably transmitted messages.

The `org.dclayer` packages contain all required classes.

Usage

Connecting to the DCL service

To use DCL in an application, a `Service` object needs to be created first. Below is an example, where `port` is an integer containing the port number the DCL service is listening on.

```
Service service = new Service(  
    port);
```

Afterwards, an `ApplicationInstance` object needs to be created for the application to be connected to the service. This works best by utilizing an `ApplicationInstanceBuilder` object, which is returned by `Service.applicationInstance()`.

```
ApplicationInstanceBuilder  
builder = service.  
applicationInstance();
```

This fluent interface can be used to set the keypair to use as the application's address, to join DCL networks and to connect the application to the service. The code below will use the `KeyPair` object referenced by `addrKeyPair` as this application's address, join the default DCL networks, register the object referenced by `listener` as the `NetworkEndpointActionListener` for the default network endpoints and connect the application to the DCL service.

A network endpoint is a pair that consists of an address and a DCL network that address has joined. Unreliably transmitted packets are sent over a DCL network and between two addresses – thus between two network endpoints on the same network. The DCL application to service protocol utilizes network endpoint slots, which are essentially numbers referring to network endpoints, to specify the origin of a message sent from the application and the destination of a message sent to the application. The DCL appli-

cation library uses `NetworkEndpointSlot` objects to define those network endpoint slots and to refer to the address and network of a network endpoint. The `NetworkEndpointSlot` object is required in every library method that requires the address and network that should be used to be specified and is passed in every callback method defined in the `NetworkEndpointActionListener` interface.

```
ApplicationInstance application  
= builder  
    .addressKeyPair(addrKeyPair)  
    .joinDefaultNetworks(listener  
    )  
    .connect();
```

The call to `connect()` will block until the TCP connection to the service is established and the application-to-service protocol is initiated. If an error occurs, a `ConnectionException` will be thrown. Otherwise, `connect()` will return a new `ApplicationInstance` object, which can be used to send unreliably transmitted packets and to initiate application channels.

Callbacks

NetworkEndpointActionListener

The `NetworkEndpointActionListener` interface defines methods for receiving callbacks upon joining of DCL networks (`onJoin()`), receipt of unreliable transmitted packets (`onReceive()`) and incoming application channel requests (`onApplicationChannelRequest()`).

ApplicationChannelActionListener

The `ApplicationChannelActionListener` interface defines methods for receiving callbacks upon successful connection (`onApplicationChannelConnected()`) and disconnection (`onApplicationchannelDisconnected()`) of an application channel.

Unreliably transmitted packets

Sending

In order to send unreliably transmitted packets, the `send()` method of an `ApplicationInstance` object needs to be called. The `send()` method takes 3 arguments: a `NetworkEndpointSlot` object describing the address and network to use, a `Data` object containing the destination address and another `Data` object containing the payload to transmit, respectively.

Receiving

Upon receipt of an unreliably transmitted packet, the `onReceive()` method of the `NetworkEndpointActionListener` assigned to the address and network the packet was received on is called and passed 3 arguments: the `NetworkEndpointSlot` object corresponding to the address and network the packet was received on, a `Data` object containing the source address and another `Data` object containing the payload received, respectively.

Note that the source address might be empty, indicating that the origin chose not to include its own address in the message.

Application channels

Initiating

To request an application channel to a specific address, the `requestApplicationChannel()` method of an `ApplicationInstance` object needs to be called and passed 4 arguments: a `NetworkEndpointSlot` object to use as the source of this application channel, a `String` used as an action identifier which the remote will receive in its `onApplicationChannelRequest` callback, a `Key` object containing the public key used as address by the remote that the application channel should be connected to and an `ApplicationChannel-`

`ActionListener` object that will receive callbacks regarding the application channel.

Accepting

When an application channel is requested, the `onApplicationChannelRequest()` of the `NetworkEndpointActionListener` object assigned to the network endpoint the application channel request was received on is called and passed 4 arguments: the `NetworkEndpointSlot` object corresponding to the network endpoint the request was received on, a `Key` object containing the public key used as address by the remote that requested the application channel, a `String` object containing the action identifier as specified by the remote and an `LLA` object containing the lower-level address (i.e., the IP address and port) of the remote requesting the application channel.

To accept the application channel request, return an `ApplicationChannelActionListener` object that will receive callbacks regarding the application channel.

To ignore the request, simply return `null`.

Usage

As soon as the application channel is successfully connected, the `onApplicationChannelConnected()` method of the `ApplicationChannelActionListener` that was either passed when calling the `requestApplicationChannel()` method or returned from the `onApplicationChannelRequest()` callback method will be called. The `onApplicationChannelConnected()` callback will be passed an `ApplicationChannel` object that refers to the established application channel.

The `ApplicationChannel` object has the following methods that can be utilized:

`getOutputStream()`

Returns a `BufferedOutputStream` object that can be used to write data which will be securely sent through

the interservice connection of the local and the remote service and then made available for the remote to read from its `InputStream` object.

Call `flush()` on the `BufferedOutputStream` object to make sure the data written is sent.

`getInputStream()`

Returns an `InputStream` object that can be used to read the data which the remote wrote into its `BufferedOutputStream` object obtained via `getOutputStream()`.

`getRemotePublicKey()`

Returns a `Key` object containing the public key used as address by the remote.

`getActionIdentifier()`

Returns a `String` object containing the action identifier that was passed to the `requestApplicationChannel()` method by the end that initiated the application channel.

`wasInitiatedLocally()`

Returns `true` if this application channel was initiated by this end, `false` otherwise.

B. Anhang 2



PEER-TO-PEER DATEISYNCHRONISATION

sicher

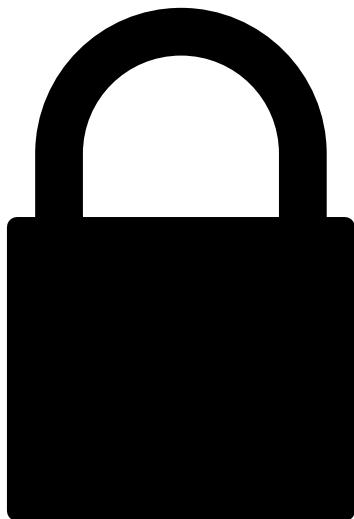
dezentral

ausfallsfrei

open source

ÜBERBLICK

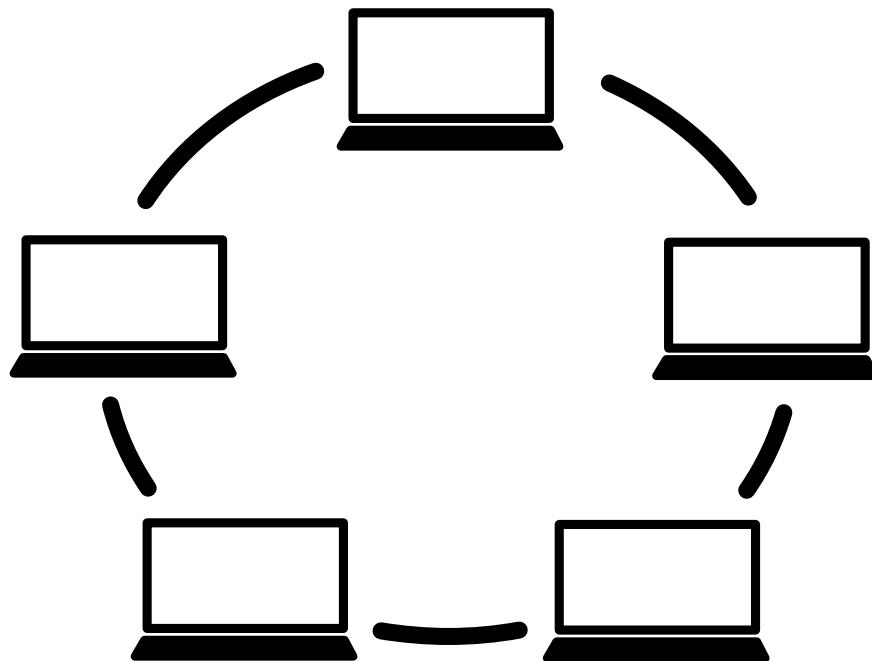
sicher



Starke Verschlüsselung

Alle Daten und Kommunikationskanäle
werden mit AES im Modus GCM
verschlüsselt. Knacken unmöglich.

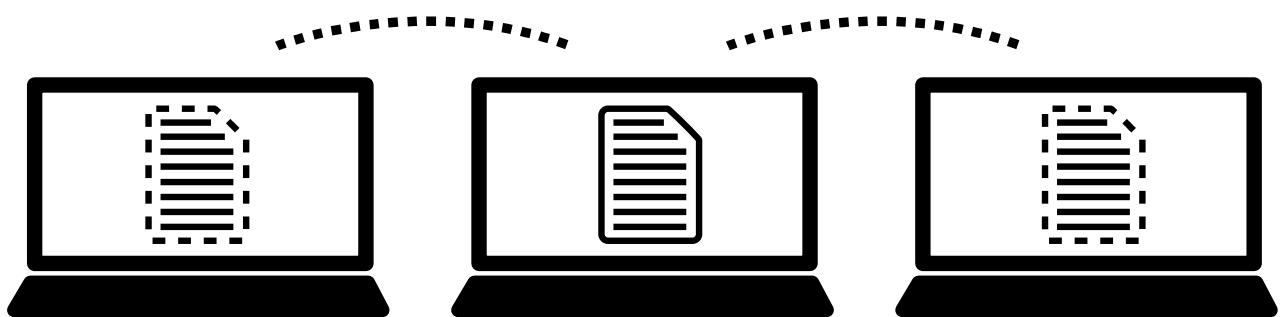
dezentral



Keine zentralen Stellen

Die Geräte kommunizieren über ein
Peer-to-Peer-Netzwerk direkt miteinander.
Es gibt keine Server oder andere zentrale
Stellen, die abgehört werden könnten.

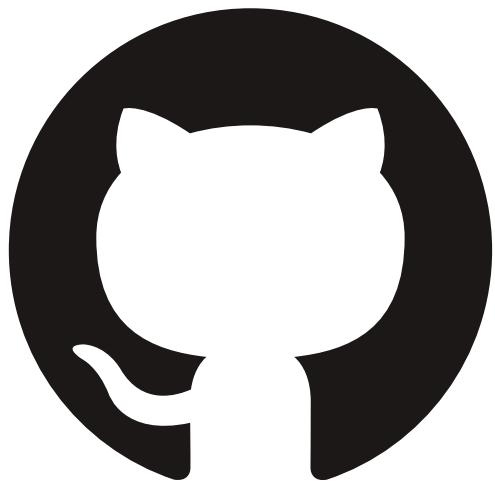
ausfallsfrei



Redundanz

Alle Dateien werden mehrmals auf verschiedenen Geräten gespeichert. Die Nicht-Erreichbarkeit der Dateien aufgrund des Ausfalls des dezentralen Systems ist praktisch unmöglich.

open source

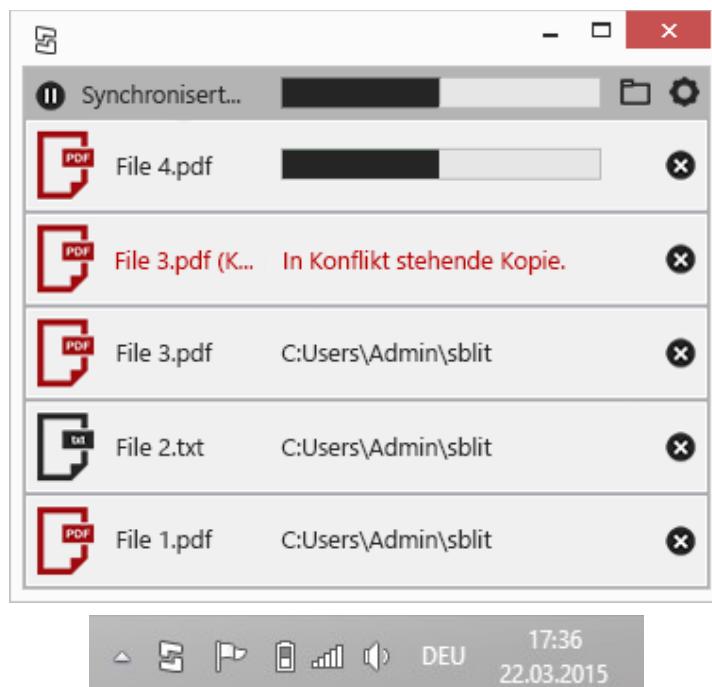


GitHub

Der Quellcode ist unter der GNU Public License in Version 3 lizenziert und auf github.com/sblit öffentlich zugänglich.

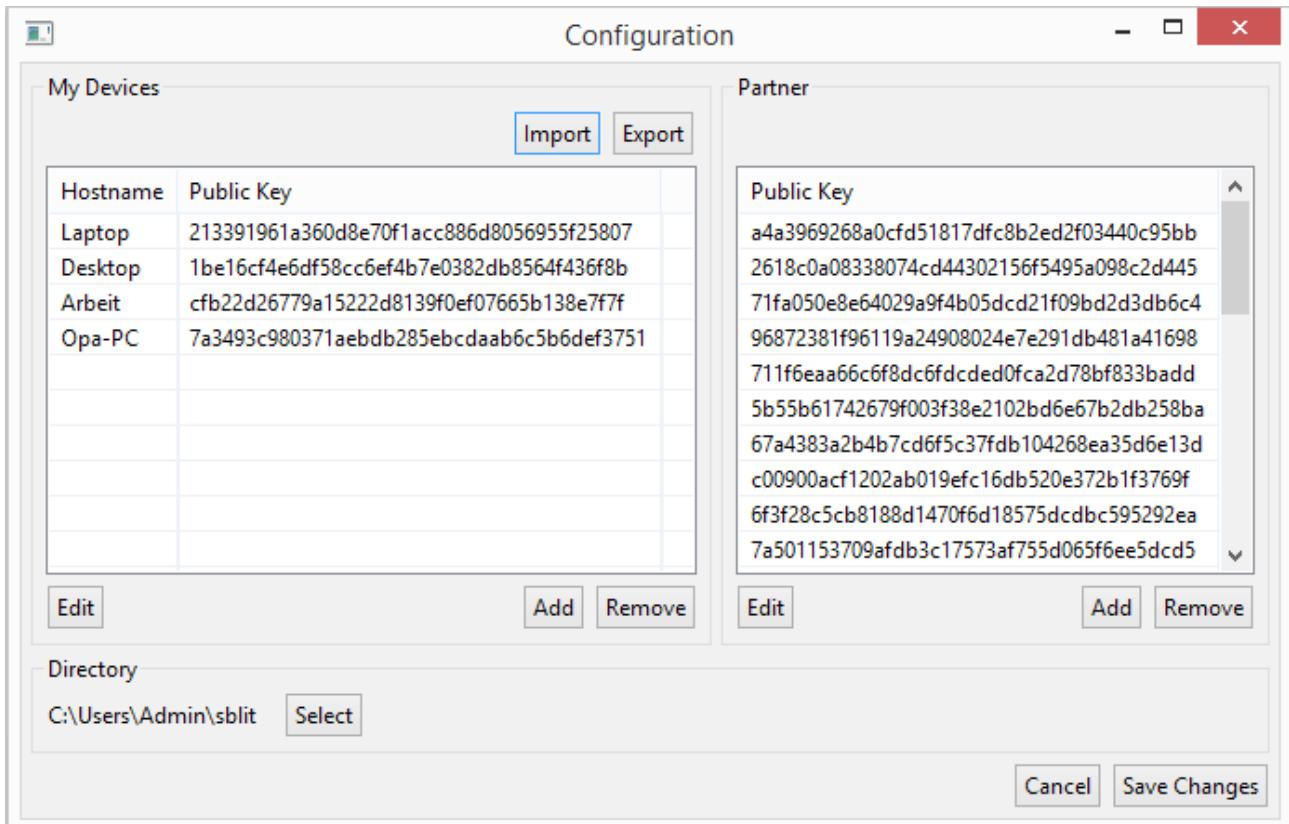
GRAPHISCHE OBERFLÄCHE

Statusübersicht



Das Hauptfenster des zurückhaltenden GUIs lässt sich mit einem Klick auf das Icon im System-Tray aufrufen. Hier werden die letzten Änderungen innerhalb des sblit-Ordners, aufgetretene Konflikte und der Fortschritt der laufenden Synchronisation angezeigt.

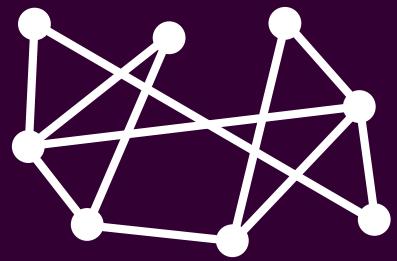
Konfiguration



Mit einem Klick auf das Zahnrad im Hauptfenster öffnet sich das Konfigurationsmenü, in dem der User die Liste der Synchronisationsgeräte und die sonst automatisch verwaltete Liste der Partnergeräte bearbeiten kann. Außerdem lässt sich hier der Pfad des zu synchronisierenden Ordners verändern.

TECHNIK

Peer-to-Peer-Netzwerk



OFFENES PEER-TO-PEER-NETZWERK

Sblit kommuniziert über ein eigens entwickeltes, offenes Peer-to-Peer-Netzwerk, den sogenannten Decentralized Communication Layer (DCL). Dieser stellt neben sblit auch Anwendungen von Fremdentwicklern flexible und vor allem sichere Kommunikationskanäle zur Verfügung.

NETZWERK AUS NETZWERKEN

DCL beherbergt dabei nicht nur ein Peer-to-Peer-Netzwerk, sondern ermöglicht es, praktisch unendlich viele Netzwerke, genauestens abgestimmt auf die Anforderungen der Anwendungen, aufzuziehen.

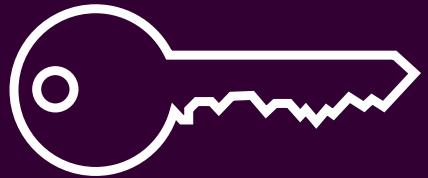
VERBINDUNGSVERSCHLÜSSELUNG

Die Kommunikation zwischen entfernten DCL-Instanzen erfolgt über UDP auf zufälligen Ports. Dabei wird die gesamte Payload der UDP-Pakete mit AES im Modus GCM verschlüsselt, was es unmöglich macht, auf den Inhalt des Datenstroms zu schließen.

APPLICATION LIBRARY

Eine Application Library zur Einbindung von DCL in Drittanwendungen bietet deren Entwicklern eine einfache Möglichkeit, Peer-to-Peer-Kommunikation umzusetzen. Statt erst ein eigenes Netzwerk aufzubauen zu müssen, wird auf die bestehende Menge an Nodes im DCL zurückgegriffen.

Adressierung mit RSA



DEZENTRALE ADRESSIERUNG

Durch die vollständige Dezentralisierung und die damit einhergehende Gleichberechtigung aller Nodes im DCL können Adressen weder von zentralen Stellen vergeben noch validiert werden. Deshalb kommt im DCL zur Adressierung das asymmetrische Verschlüsselungsverfahren RSA zum Einsatz. Jeder Node generiert beim Start ein RSA-Schlüsselpaar, dessen öffentlicher Schlüssel als Adresse verwendet wird.

VALIDIERUNG VON ADRESSEN

Solche Adressen können gänzlich ohne zentrales Organ auf ihre Echtheit überprüft werden, indem der zu überprüfende Node mit seinem privaten Schlüssel eine Signatur von Daten anfertigt, die vom überprüfenden Node generiert wurden. Der überprüfende Node kann diese Signatur anschließend mit der Adresse, also dem öffentlichen Schlüssel des zu überprüfenden Nodes validieren und somit sichergehen, dass seine Adresse echt ist.

ROUTING

Zum Finden von Wegen durch das DCL-Netzwerk kommt ein Algorithmus zum Einsatz, der dem des Kademlia-Modells sehr ähnlich ist. Eine Nachricht wird dabei schrittweise von einem Node zu dem Node weitergeleitet, dessen Adresse die geringste Differenz zur Zieladresse aufweist, bis die Nachricht schließlich ihr Ziel erreicht. Um die Länge von Adressen zu verringern und dadurch das Routing und die Kommunikation effizienter zu machen, werden die eigentlichen Adressen durch Hashen der öffentlichen Schlüssel erzeugt. Als Hashalgorithmus wird dabei SHA-1 verwendet, wodurch sich Adressen mit einer Länge von 20 Bytes ergeben.

Synchronisation



VERFÜGBARKEIT

Damit die Änderungen auch synchronisiert werden können, wenn immer nur eines der eigenen Geräte gleichzeitig online ist, werden die Änderungen auf Geräten anderer Nutzer zwischengespeichert. Diese Geräte werden Partnergeräte genannt.

DELTA-DATEN

Sblit synchronisiert Delta-Daten. Unter Delta-Daten kann man sich die Änderungen an einer Datei vorstellen. Bei einer Änderung werden diese auf den Partnergeräten gespeichert und wieder gelöscht, sobald die Änderung auf alle eigenen Geräten übertragen wurde.

VERSCHLÜSSELUNG

Sblit verschlüsselt die Dateien mithilfe eines zufällig generierten AES-256-Schlüssels. Beim Hinzufügen eines neuen Gerätes kann dieser exportiert und z.B. über einen USB-Stick auf das neue Gerät kopiert werden. Angenommen, man verwendet den stärksten Supercomputer der Welt, so benötigt man durchschnittlich 10^{56} Jahre, um den Schlüssel durch ausprobieren zu knacken. Das ist in etwa das 10^{46} -fache Alter des Universums.

ERKENNEN VON ÄNDERUNGEN

Damit Änderungen sofort erkannt und synchronisiert werden können, verwendet sblit das Watchservice. Dieses ist eine Schnittstelle zum Dateisystem, welche benachrichtigt wird, sobald sich eine Datei in einem bestimmten Ordner ändert.



Synchronisationskonflikte

KONFLIKT

Konflikte treten auf, wenn ein und dieselbe Datei gleichzeitig auf zwei verschiedenen Geräten geändert wird. Dabei entstehen zwei unterschiedliche Versionen dieser Datei.

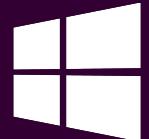
ERKENNUNG

Dieses Problem erkennt sblit mit einer internen, Hash-basierten Versionierung. Dabei wird zu jeder Datei ein Versionsverlauf gespeichert. Bei einer Änderung wird immer der zur Datei passende Versionsverlauf mitgeschickt. Stimmt der letzte Hash im lokalen Versionsverlauf mit einem beliebigen Hash im mitgeschickten Versionsverlauf überein, tritt kein Konflikt auf. Enthält der empfangene Versionsverlauf jedoch nicht den aktuellsten Hash im lokalen Versionsverlauf, wurde die Datei geändert, bevor diese synchronisiert werden konnte. Anders gesagt: Ein Konflikt ist aufgetreten.

LÖSUNG

Um einen Konflikt zu lösen, wird dem Namen der Datei auf dem aktuellen Gerät das Suffix „**(Konflikt 1)**“ hinzugefügt und die empfangene Datei unter dem Originalnamen gespeichert. Falls bereits eine gleichnamige Datei mit dem Suffix „**(Konflikt 1)**“ existiert, wird die Nummer entsprechend erhöht. Diese Konfliktdatei wird dann ebenfalls auf das andere Gerät synchronisiert.

Plattformunabhängigkeit



JAVA

Sowohl sblit als auch die Unterkomponenten DCL und die Application Library sind in Java geschrieben, um den reibungslosen Einsatz auf allen Plattformen gewährleisten zu können.

BOUNCY CASTLE

Um die notwendigen kryptographischen Funktionen auf den unterschiedlichen Plattformen uneingeschränkt verwenden zu können, benutzt sblit die freie Bibliothek Bouncy Castle, die auch bei Googles mobilem Betriebssystem Android zum Einsatz kommt.

SWT

Damit der Plattform-spezifische Look auf den verschiedenen Betriebssystemen nicht verloren geht, verwendet sblit SWT, eine Programmbibliothek für graphische Oberflächen. SWT benutzt die OS-nativen Widgets. Dadurch haben die GUIs auf allen Plattformen das für das jeweilige Betriebssystem typische Aussehen, obwohl überall der gleiche Programmcode ausgeführt wird.

TEAM



Andreas Novak

Graphische Oberfläche, Website



Nikola Szucsich

Synchronisationsanwendung



Martin Exner

Kommunikationsschicht, Projektleitung

FORTSCHRITT

Fortschritt

per 22.03.2015



UMGESETZT

Online-Synchronisation
Benutzeroberfläche
Dateiverschlüsselung
Verbindungsverschlüsselung
Adressierung
Routing
Application Library

AUSSTEHEND

Partnerschaften (Offline-Synchronisation)
Effizienter Netzwerkeinstieg
Verbinden der einzelnen Komponenten

C. Anhang 3

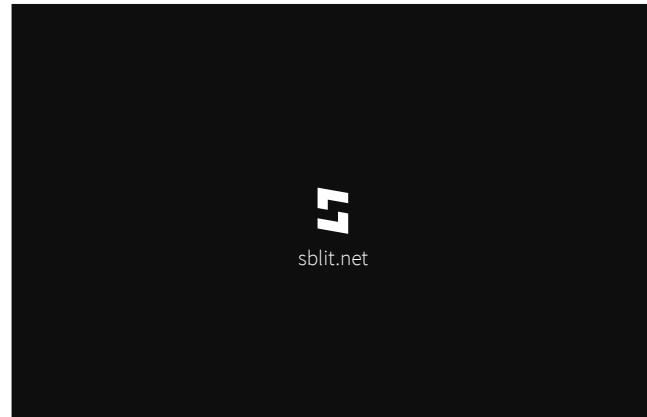


Abbildung C.1.: Vorderseite der Visitenkarte

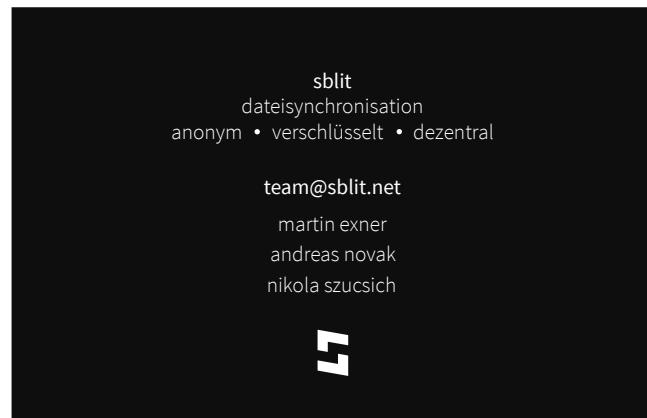


Abbildung C.2.: Rückseite der Visitenkarte

Literaturverzeichnis

- [1] *Hash function* – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Hash_function, Abruf: 2015-03-06
- [2] *How many RSA keys before a collision?* – Cryptography Stack Exchange. <http://crypto.stackexchange.com/a/2559>, Abruf: 2015-02-24
- [3] *Keylength – BSI Cryptographic Key Length Report for Digital Signature (2015)*. <http://www.keylength.com/en/8/>, Abruf: 2015-04-05
- [4] *RSA (cryptosystem)* – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/RSA_%28cryptosystem%29, Abruf: 2015-03-05
- [5] *Transmission Control Protocol* – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Transmission_Control_Protocol, Abruf: 2015-04-01
- [6] *User Datagram Protocol* – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/User_Datagram_Protocol, Abruf: 2015-04-01

Akronyme

AES Advanced Encryption Standard. 20, 21, 79, 129, *Glossar*: Advanced Encryption Standard

BMCP Basic Management Channel Protocol. 21–24, 26, 27, 128, *Glossar*: Basic Management Channel Protocol

CRISP Common Routed Interservice Protocol. 49–51, *Glossar*: Common Routed Interservice Protocol

DCL Decentralized Communication Layer. 9–16, 19, 25, 37, 48–50, 52, 58, 65, 75, 127–132, *Glossar*: Decentralized Communication Layer

GCM Galois/Counter Mode. 20, 21, 79, *Glossar*: Galois/Counter Mode

GUI Graphical User Interface. 85, 86, *Glossar*: Graphical User Interface

LLA Lower Level Address. 28–30, 34, 43, 49, 50, *Glossar*: Lower Level Address

MVC Model-View-Controller-Design. 84, *Glossar*: Model-View-Controller-Design

NAT Network Address Translation. 17, 18, 49, 130, *Glossar*: Network Address Translation

SWT Standard Widget Toolkit. 84, 85, *Glossar*: Standard-Widget-Toolkit

TCP Transmission Control Protocol. 37, 127, *Glossar*: Transmission Control Protocol

UDP User Datagram Protocol. 20, 130, *Glossar*: User Datagram Protocol

Glossar

Action Identifier String, der in Anfragen übertragen wird und Auskunft über deren Grund geben soll. 43, 49, 50

Address Im Quellcode von DCL definierte Klasse zur Speicherung von öffentlichen Schlüsseln anderer Teilnehmer. 13, 16, 129

Address Slot Zahl zur Referenzierung von auf einem DCL-Service gehosteten Adressen. 25, 27, 28, 30, 32, 33, 35, 128

Advanced Encryption Standard Standard-Verschlüsselungsverfahren für Links. 20, 125, *Siehe:* Link

Application Channel Ein verlässlicher Übertragungskanal zur Kommunikation zwischen zwei Instanzen einer auf dem DCL aufbauenden Anwendung. 28, 35, 37, 42, 43, 47–50, 66, 75–77, 127

Application Channel Slot Zahl zur Referenzierung von Application Channels in Inter-service Channels. 28, 35, 42, 43, 47

Application-to-Service-Protokoll Das zur Kommunikation zwischen einer Anwendung und einem DCL-Service verwendete Protokoll. 35, 37–48, 127

Application-to-Service-Verbindung Verbindung über TCP zwischen einer Anwendung und einem DCL-Service, auf der mittels Application-to-Service-Protokoll kommuniziert wird. 37, 48, 49, 130

Asymmetrisches Verschlüsselungsverfahren Verschlüsselungsverfahren, bei dem zur Entschlüsselung einer Nachricht ein anderer Schlüssel verwendet wird, als zur Verschlüsselung. 11, 132, *Siehe:* RSA

Authenticity Request Eine Nachricht zur Anforderung einer Challenge, um die Authentizität des Gegenübers sicherzustellen. 66, 67, 76, 127

Authenticity Response Die Antwort auf den Authenticity Request, um die eigene Authentizität zu beweisen. 67, 76

Basic Management Channel Protocol Standardmäßig verwendetes Management Channel Protocol zur Verwaltung von Links. Siehe 2.7.4 Basic Management Channel Protocol, Seite 21. 21, 125

Channel Getrennter Datenkanal eines Links. Siehe 2.7.3 Channels, Seite 20. 20, 21, 24, 25, 127–131

- Channel Block Status Report** Statusinformation über die empfangenen Data Identifiers eines Channels, anhand der fehlende Pakete erkannt und erneut übertragen werden können. 25, *Siehe:* Channel
- Channel Identifier** Zahl zur eindeutigen Identifizierung eines Channels auf einem Link. 20, 24–27
- Circle Network** Standard Netzwerk im DCL, Network Type Identifier `org.dclayer.circle sha1/20`. 10–12, 14, 39, 128, *Siehe:* Netzwerk
- CircleNetworkType** Im Quellcode von DCL definierte Klasse zur Darstellung des Network Type des Circle Network. 12, 13, 15, 16, *Siehe:* Network Type
- Cloudspeicher** Externer Speicherplatz in einer Filecloud. 2, 3
- Common Routed Interservice Protocol** Protokoll, das netzwerkübergreifend zur Kommunikation zwischen nicht direkt miteinander verbundenen DCL-Services verwendet wird. Siehe 2.10.4 CRISP, Seite 50. 49, 50, 125
- Connection Base** Berechtigungslevel für einen Address Slot eines Interservice Channel. 27–30, 32, 33, 35
- Crypto Challenge** Anforderung, mit dem einem bekannten öffentlichen Schlüssel zugehörigen privaten Schlüssel Daten zu signieren, um so den Besitz des gesamten Schlüsselpaares zu beweisen. 28, 30, 32, 37
- Crypto Initialization Method** Art der Verschlüsselung eines mit BMCP verwalteten Links und Methode zur Initialisierung dieser Verschlüsselung. 21–23, 128, *Siehe:* Basic Management Channel Protocol
- Crypto Initialization Method Identifier** Zahl zur eindeutigen Kennung einer Crypto Initialization Method. 22, *Siehe:* Crypto Initialization Method
- Data** Im Quellcode von DCL definierte Klasse zur Speicherung von Binärdaten. 13, 16, 17, 65, 129
- Data Identifier** Fortlaufende Zahl zur eindeutigen Identifizierung eines über einen Channel eines Links übertragenen Datenblocks und zur Angabe der Reihenfolge dieses Datenblocks im Datenstrom des Channels. 20, 23–26, 127
- DCL-Service** Ein Teilnehmer des DCL. Kann mehrere Adressen in mehreren unterschiedlichen Netzwerken hosten. 17, 19, 20, 25, 27–30, 32–35, 37–40, 42, 43, 47–50, 127–131, *Siehe:* DCL
- Decentralized Communication Layer** Die auf einem dezentralen Peer-to-Peer-Netzwerk basierende Kommunikationschicht, die sblit zur Kommunikation benutzt. 9, 125
- Device Refresh Message** Eine Nachricht, die eine aktualisierte Version von Partnergeräten oder eigenen Geräten enthält. 68, 71, 77
- Digest Length** Die Länge der Hashwerte eines Hashalgorithmus. 10, 13

Endpunkt Eine auf dem lokalen bzw. einem direkt verbundenen DCL-Service gehostete Adresse in einem bestimmten Netzwerk. 16, 17, 25, 37, 129, 131, *Siehe:* Nexthops

File Delete Message Eine Nachricht, die dazu dient, Geräte über eine Löschung einer Datei zu informieren. 68, 71, 72, 75, 77, 131

File Delete Partner Message Eine Nachricht, die dazu dient, Dateien von Partnergeräten zu löschen. 75

File Message Die eigentliche Übertragung einer Datei. 68, 69, 77, 131

File Request Die Anfrage an das Gegenüber, ob eine Datei benötigt wird. 64, 67–69, 72, 76, 77, 131

File Response Die Antwort auf einen File Request, der eine Datei akzeptiert oder verweigert. 67–69, 73, 77, 131

Filecloud Externer Speicher im Internet, auf dem Dateien gespeichert werden können. Meistens bestehend aus einem oder mehreren Servern. 4–8, 128

FlexNum-Component Packet Component zur Übertragung einer bis zu 64 Bits großen Ganzzahl. Siehe 2.11 FlexNum, Seite 50. 20, 22, 24, 26–36, 38, 40–48, 50–54, 68, 70, 72–74, 76

Flusskontrolle Einschränkung der über einen Kommunikationskanal übertragenen Menge an Daten zur Vermeidung von Verstopfungen. 21, 25

ForwardDestination Im Quellcode von DCL definierte Klasse zur Speicherung eines verbundenen Endpunkts, an den eine Nachricht weitergeleitet werden kann, inklusive dessen Address- und InterserviceChannel-Objekten. 16, 131, *Siehe:* Endpunkt

Galois/Counter Mode Standard-Betriebsmodus für die Verschlüsselung von Links mit AES. 20, 125

Graphical User Interface Eine grafische Oberfläche für Benutzer, um eine Anwendung komfortabel bedienen zu können. 125

Hash Im Quellcode von DCL definierte Klasse zur Anwendung von Hashalgorithmen auf Data-Objekte. 13

Hashalgorithmus Algorithmus, der eine große Eingabemenge auf eine kleinere Zielmenge (Hashwert) abbildet [1]. 10, 12, 13, 128, 129, 132

Hashwert Ergebnis der Anwendung eines Hashalgorithmus. 13, 128, 129

Ignore Data Daten, die als Präfix für NAT-Traversal-Pakete verwendet werden können, da Pakete, die diese Daten enthalten, vom Empfänger ignoriert werden. 43, 49, 50, *Siehe:* NAT-Traversal-Paket

Interservice Channel Channel eines Links zwischen zwei DCL-Services, innerhalb dessen mit dem Interservice-Protokoll kommuniziert wird. 25, 27, 37, 48, 49, 127, 128, 130

- Interservice-Protokoll** Das zur Kommunikation zwischen zwei direkt verbundenen DCL-Services verwendete Protokoll. 25, 27–36, 129
- InterserviceChannel** Im Quellcode von DCL definierte Klasse zur Verwaltung einer Verbindung zwischen zwei DCL-Services. 16, 129, *Siehe:* Interservice Channel
- KeyComponent** Basis für Packet Components zur Übertragung von Schlüsseldaten. 53, 54
- Kommunikationsschicht** Eine Schicht im netzwerktechnischen Sinne, die zur Kommunikation verwendet wird. 9, 66, 128, *Siehe:* Decentralized Communication Layer
- Link** Verlässlicher und verschlüsselter Übertragungskanal basierend auf UDP. Basis für Verbindungen zwischen DCL-Services. Siehe 2.7 Links, Seite 20. 20–23, 25, 48, 127–131
- LogFile** Die Datei, in der der Versionsverlauf einer Datei steht. Außerdem werden hier die Geräte, auf denen die Datei schon vorhanden ist, gelistet. 56, 59, 63, 64, 66, 76, 77
- Lower Level Address** Die Adresse eines DCL-Service auf der Netzwerkschicht unter DCL, de facto die IP-Adresse und Port, unter der ein DCL-Service erreichbar ist. 125
- Management Channel** Channel eines Links, der zur Übertragung von Nachrichten zur Verwaltung des Links genutzt wird. 20, *Siehe:* Channel
- Management Channel Protocol** Protokoll zur Verwaltung eines Links. 21, 23, 127, *Siehe:* Basic Management Channel Protocol
- Model-View-Controller-Design** Ein Programmier-Pattern, bei dem Datenzugriff und Anwendungslogik von der grafischen Darstellung getrennt sind.. 125
- NAT Hole Punching** Verfahren zum NAT-Traversal, das in der Implementierung des DCL-Service angewandt wird. 19, *Siehe:* NAT-Traversal
- NAT-Gerät** Gerät, das Network Address Translation durchführt. 18, 19, 49, 130
- NAT-Traversal** Umgehung von NAT-Geräten, um Peer-to-Peer-Verbindungen aufzubauen zu können. 18, 20, 49, 130
- NAT-Traversal-Paket** Paket, das im Zuge von NAT-Traversal gesendet wird. 43, 49, 50, 129, *Siehe:* NAT-Traversal
- Network Address Translation** Auf NAT-Geräten durchgeföhrte Übersetzung von Netzwerkadressen. Siehe 2.6 NAT-Traversal, Seite 17. 17, 18, 125, 130
- Network Endpoint** Eine auf einer Application-to-Service-Verbbindung bekanntgegebene Adresse in einem Netzwerk, dem die Anwendung mit dieser Adresse beigetreten ist. 37, 39, 130
- Network Endpoint Slot** Zahl zur Referenzierung von Network Endpoints auf Application-to-Service-Verbbindungen. 37, 39, 40, 42, 43

Network Packet Geroutete Nachricht innerhalb eines Netzwerks. 27, 28, 35

Network Slot Zahl zur Referenzierung von Netzwerken, denen ein DCL-Service auf einem Interservice Channel mit einer oder mehreren Adressen beigetreten ist. 27, 28, 33, 35

Network Type Beschreibt ein Netzwerk innerhalb des DCL samt Adresskonzept und Routingverfahren. 10, 13, 39, 128, 131, *Siehe:* Netzwerk

Network Type Identifier Der eindeutige Name eines Network Types. 10, 27, 33, 128, *Siehe:* Network Type

Netzwerk Ein Peer-to-Peer-Netzwerk innerhalb des DCL. 10, 19, 25, 27, 28, 30, 33–35, 37, 39, 40, 49, 50, 128, 130, 131, *Siehe:* Circle Network

Netzwerkintegration Das Aufbauen von Verbindungen mit anderen DCL-Services, um optimal in das Routing eines Netzwerks eingebunden zu sein. 50

Nexthops Im Quellcode von DCL definierte Klasse zur Speicherung von verbundenen Endpunkten in der Form von `ForwardDestination`-Objekten. 16, 17

Packet Component Teil einer Message, beispielsweise Ganzzahl oder String. Siehe 2.11 Packet Components, Seite 50. 50, 53, 129, 130

Partner File Delete Message Eine File Delete Message an ein Partnergerät (siehe 3.5.2 Partnergeräte, Seite 79) oder von einem Partnergerät. 72, 75

Partner File Message Eine File Message an ein Partnergerät (siehe 3.5.2 Partnergeräte, Seite 79) oder von einem Partnergerät. 72, 73, 78

Partner File Request Ein File Request an ein Partnergerät (siehe 3.5.2 Partnergeräte, Seite 79) oder von einem Partnergerät. 72, 73, 77, 78

Partner File Response Ein File Response an ein Partnergerät (siehe 3.5.2 Partnergeräte, Seite 79) oder von einem Partnergerät. 72, 73, 78

Partnergerät Ein Gerät eines anderen sblit-Nutzers, mit dem eine Partnerschaft besteht. 5–8, 55, 75, 77–79

Partnerschaft Um Dateien/Dateiblöcke extern speichern zu können, muss eine sogenannte Partnerschaft mit anderen Geräten eingangen werden, bei der man Dateiblöcke auf den jeweilig anderen Geräten speichert, während man selber Speicherplatz für diese Partner freigibt, in dem deren verschlüsselte Dateiblöcke gespeichert werden. 5

Peer-to-Peer-Netzwerk Ein Kommunikationsnetzwerk, in dem die einzelnen Kommunikationspartner im Gegensatz zu einer klassischen Client-Server-Infrastruktur direkt miteinander kommunizieren. 5, 9, 10, 14, 128, 131

Prix Ars Electronica Ist der traditionsreichste Medienkunstwettbewerb wird gemeinsam von der Ars Electronica Linz GmbH und dem ORF Oberösterreich in Zusammenarbeit mit dem OK Offenes Kulturhaus Oberösterreich und dem Brucknerhaus Linz veranstaltet. . 90

Protocol Identifier Eindeutige Kennung des Protokolls, über das auf einem Channel eines Links kommuniziert wird. 23–25, 27

Remote Key Schlüssel, der lokal nicht bekannt ist und zu dessen besitzender Anwendung eine Verbindung besteht, über die Verschlüsselungs- und Entschlüsselungsanfragen weitergeleitet werden. Siehe 2.9.4 Remote Keys, Seite 37. 37, 38

RSA Asymmetrisches Verschlüsselungsverfahren, benannt nach und erstmals veröffentlicht von Ron Rivest, Adi Shamir and Leonard Adleman [4]. 10–12, 21, 53, *Siehe:* Asymmetrisches Verschlüsselungsverfahren

sblit Dateisynchronisationsanwendung. 67–76

SHA-1 Secure Hash Algorithm. Im DCL standardmäßig verwendeter Hashalgorithmus. 10, *Siehe:* Hashalgorithmus

Standard-Widget-Toolkit Ein freies Framework für grafische Oberflächen, entwickelt von Eclipse.. 125

Synchronisationskonflikt Wenn eine synchronisierte Datei von zwei Rechnern bearbeitet wird, ohne, dass sie in der Zwischenzeit synchronisiert werden konnte. Es existieren dann zwei Versionen der gleichen Datei. Man spricht dann von einem Konflikt.. 4, 5, 82

Synchronisationspartner Gerät mit dem der sblit-Ordner synchronisiert wird. 5–7, 81, 82

Transmission Control Protocol Verbindungsorientiertes, verlässliches Übertragungsprotokoll aus der Internet protocol suite [5]. 37, 125

User Datagram Protocol Verbindungsloses Übertragungsprotokoll aus der Internet protocol suite [6]. 20, 125

WatchKey Ein Objekt, das vom WatchService erstellt wird, wenn sich etwas in dem zu überwachenden Ordner ändert. 60

WatchService Eine Schnittstelle zum Filesystem, welche benachrichtigt wird, wenn sich eine Datei in einem bestimmten Ordner ändert. 59, 60, 76, 77, 132

What-you-see-is-what-you-get-Editor Ein Editor, der während der Bearbeitung eines Dokuments, gleichzeitig das Ergebnis grafisch darstellt. Beispiele wären der Windowbuilder bei SwingOffice-Programme wie Microsoft Word oder Libreoffice Writer.. 84