



# Signalbot

Fachübergreifende Dokumentation eines Community-Bots  
für die Nachrichtenapp Signal

**Co-Autor:** Frederic Aust

015 201 285

Algorithmen und Datenstrukturen

**Co-Autor:** Philip Maas

015 200 898

Treiberentwicklung, Echtzeit- und Betriebssysteme

**Erstprüfer:** Prof. Dr. rer. nat. Peter Gerwinski

**Zweitprüfer:** M. Sc. Benedikt Wildenhain

**Abgabedatum:** 21. 10. 2021

## **Abstract**

In dieser Dokumentation wird die Vorgehensweise zur Erstellung eines Signal Community Bots dargestellt. Dieser verwendet das Signal Command Line Interface, um Nachrichten über einen eigenen Signal-Account zu empfangen und zu versenden. Es wurden diverse, von Nutzer\*innen erfragte Kommandos eingebaut, die sich in die Kategorien statische Texte, APIs, Crawler, Datenbanken und Diverses einordnen lassen. Alle Funktionen des Bots werden bereits von mehreren Gruppen und ehrenamtlichen Organisationen verwendet, weshalb es auch ein Berechtigungssystem gibt.

This documentation describes the procedure for creating a Signal community bot. It uses the Signal Command Line Interface to receive and send messages from it's own Signal account. Various user-requested commands have been included, which can be categorized as static texts, APIs, crawlers, databases, and miscellaneous. All of the bot's functions are already used by several groups and volunteer organizations, which is why there is also an authorization system.

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>i</b>
<b>Abkürzungsverzeichnis</b>	<b>iii</b>
<b>1 Einleitung</b>	<b>2</b>
1.1 Motivation . . . . .	2
1.2 Zielsetzung . . . . .	2
1.3 Signal . . . . .	3
1.3.1 X3DH . . . . .	3
1.3.2 Double Ratchet . . . . .	6
1.3.3 XEdDSA und VEdDSA . . . . .	10
1.3.4 Sesame . . . . .	11
1.4 Signal-CLI . . . . .	12
1.4.1 Installation . . . . .	12
1.4.2 Accounterstellung . . . . .	14
1.4.3 Deamon . . . . .	16
<b>2 Funktionalitäten und Module</b>	<b>19</b>
2.1 Echos und Statische Texte . . . . .	19
2.2 Programmierschnittstellen . . . . .	20
2.3 WebCrawler . . . . .	21
2.4 Zeitgesteuerte Events . . . . .	25
2.5 Datenbanken . . . . .	26
2.6 M2M-Kommunikation . . . . .	27
2.7 Weitere Funktionalitäten . . . . .	30
2.7.1 Berechtigungssystem . . . . .	30
2.7.2 Formelparser . . . . .	30
2.7.3 Lockdownchallenge . . . . .	31
2.8 Nicht umgesetzte Funktionalitäten . . . . .	32
<b>3 Fazit und Ausblick</b>	<b>33</b>
3.1 Persönliches Fazit . . . . .	33
3.1.1 Frederic Aust . . . . .	33

3.1.2 Philip Maas . . . . .	34
3.2 Ausblick . . . . .	35
<b>Abbildungsverzeichnis</b>	<b>I</b>
<b>Literatur</b>	<b>II</b>

# Abkürzungsverzeichnis

<b>AD</b>	Associated data
<b>API</b>	Application Programming Interface - Programmierschnittstelle
<b>CAPTCHA</b>	completely automated public Turing test to tell computers and humans apart
<b>CSV</b>	Comma/Character-separated values
<b>CLI</b>	Command Line Interface
<b>Dict</b>	Dictionary
<b>EK</b>	Ephemeral key
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IDE</b>	Integrated Development Environment
<b>IK</b>	Identity key
<b>IMS</b>	IMS Messsysteme GmbH
<b>JRE</b>	Java Runtime Environment
<b>JSON</b>	JavaScript Object Notation
<b>JS</b>	JavaScript
<b>KDF</b>	Key-Derivation-Function
<b>M2M</b>	Machine to Machine
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>OPK</b>	One-time prekey
<b>PDF</b>	Portable Document Format

<b>PK</b>	Public key
<b>PoC</b>	Proof of Concept
<b>REST</b>	Representational State Transfer
<b>Sesame</b>	Session Management for Asynchronous Message Encryption
<b>SK</b>	Secret key
<b>SPK</b>	Signed prekey
<b>SVG</b>	Scalable Vector Graphics
<b>td</b>	table data
<b>URL</b>	Uniform Resource Locator
<b>X3DH</b>	Extended Triple Diffie-Hellman

# 1 Einleitung

Dieses Kapitel soll die Intention, Zielsetzung und vorbereitende Schritte des Projektes darstellen. Zudem soll die Funktionsweise der Nachrichtenapp Signal beleuchtet werden.

## 1.1 Motivation

Im Februar 2021 haben viele Personen als Reaktion auf den jüngsten Datenschutzskandals der Echtzeitkommunikationsanwendung Whatsapp [1] [2] [3] [4] einen Umzug auf die Nachrichtenapp Signal unternommen. Diese wirbt mit „einem unerwarteten Fokus auf Privatsphäre“ [5], ist spenden-finanziert und wird von Datenschutzaktivisten wie Edward Snowden empfohlen [6]. Seit Februar 2020 wird der Messenger für die Mitglieder der EU-Kommission empfohlen [7]. Der Funktionsumfang ist vergleichbar zu den populären Konkurrenzprodukten. Tatsächlich soll es, ähnlich zum Nachrichtensofortversanddienstes Telegram, möglich sein automatisierte Kommunikation zu realisieren. Diese Dokumentation soll sich damit beschäftigen einen Bot für Signal zu entwickeln, der von verschiedenen Gruppen und Privatpersonen genutzt wird.

## 1.2 Zielsetzung

Ziel des Projektes ist es einen Bot für die Nachrichtenapp Signal zu realisieren. Dafür muss getestet werden, wie man über den Signal-Dienst automatisiert Nachrichten per Python unter Linux empfangen und versenden kann. Dem Bot soll ein breites Kommandospektrum zur Verfügung stehen und unterschiedliche Technologien verwenden. Alle Kommandos werden sofort nach der Implementierung durch ein Publikum von circa 40 Personen getestet. Dabei können Änderungs- oder Featurewünsche zu jeder Zeit geäußert werden. Der Bot soll intuitiv bedienbar sein. Alle neuen Benutzer\*innen bekommen als Startpunkt nur den `.help`-Befehl mitgeteilt, sodass die restliche Funktionalität selbst entdeckt werden kann.

## 1.3 Signal

Signal ist ein kostenfreier quelloffener Messenger der gleichnamigen Stiftung und besticht durch Sicherheit, sowie Datenschutz. Die Signal Technology Foundation [8] ist eine gemeinnützige US-Amerikanische Stiftung, welche 2013 gegründet wurde. Diese finanziert sich durch Spenden. Dementsprechend verdient Signal im Gegensatz zu beispielsweise Whatsapp kein Geld durch den Einsatz von Werbung oder mit dem Verkauf von Daten der Nutzer\*innen. Die Kommunikation wird nach aktuellstem Stand der Technik [9] [10] [11] sicher Ende-zu-Ende verschlüsselt und nur auf den Endgeräten gespeichert. Nach Aussage des Gründers Moxie Marlinspike in der New York Times [12] werden auf den Signal-Servern lediglich das Datum der Accounterstellung, wann sich der User das letzte Mal mit dem Server verbunden hat und temporär noch nicht zugestellte verschlüsselte Nachrichten gespeichert. Es wird nicht analysiert Wer mit Wem kommuniziert oder was Inhalt der Gespräche ist. Daher stellt es kein Problem dar, dass für die Server Anbieter wie Amazon oder Google verwendet werden. In den Abschnitten 1.3.1 bis 1.3.4 soll die Funktionsweise von Signals Verschlüsselung erläutert werden.

### 1.3.1 X3DH

*Extended Triple Diffie-Hellman (X3DH)* steht für die dreifache Ausführung des Diffie-Hellman Algorithmus mit verschiedenen Schlüsselkombinationen [13], um schlussendlich einen *Secret key (SK)* zu berechnen. Für alle verwendeten Schlüsselpaare werden entweder die Funktionen X25519 oder X448 verwendet, welche beide auf elliptischen Kurven basieren. Dabei werden Schlüsselpaare von den in 1.3.3 beschriebenen Verfahren verwendet.

### Diffi-Hellman Schlüsselaustausch

Der Algorithmus ist nach den Erfindern Whitfield Diffie und Martin Hellman benannt [14]. Dieser wurde 1976 veröffentlicht und ermöglicht es einen Schlüssel zwischen zwei Gesprächsteilnehmer\*innen auszutauschen ohne diesen zu übermitteln. Für das folgende Beispiel werden die teilnehmenden Personen Alice und Bob genannt.

1. Für den Schlüsselaustausch einigen sich Beide auf eine große Primzahl  $g$  und eine maximale Schlüssellänge  $n$ , welche aktuell üblicherweise entweder 2048 Bit oder besser 4096 Bit groß ist. Diese Zahlen müssen nicht geheimgehalten werden.



2. Anschließend wählen beide Teilnehmer\*innen eine Zahl ( $a$  für Alice,  $b$  für Bob), welche zwischen 0 und  $n$  liegt. Dies ist der private Schlüssel und wird von Beiden geheim gehalten.
3. Der öffentliche Schlüssel wird mit  $A = g^a \bmod(n)$  für Alice und  $B = g^b \bmod(n)$  für Bob berechnet.
4. Die öffentlichen Schlüssel( $A, B$ ) geben die beidem Teilnehmer\*innen dem jeweils anderen bekannt.
5. Zuletzt berechnen Beide jeweils mit  $B^a \bmod(n)$  beziehungsweise  $A^b \bmod(n)$  den gemeinsamen Schlüssel.

### Erster Gesprächsaufbau

Damit eine Kommunikation selbst dann aufgebaut werden kann, wenn Bob offline ist, hinterlegt er bei der Registrierung auf dem Signal Server seinen *Identity key (IK)*, einen *Signed prekey (SPK)* und mehrere *One-time prekey (OPK)*. Sollten diese nahezu aufgebraucht sein, wird Bob aufgefordert neue Einmalschlüssel auf dem Server zu hinterlegen. Für den Kommunikationsaufbau erzeugt Alice im Vorfeld einen *Ephemeral key (EK)*, welcher für jede weitere Nachricht neu generiert wird.

Bei dem X3DH werden die folgenden Schlüssel beim ersten Gesprächsaufbau von Alice mit Bob eingesetzt:

- $IK_A$  (Alices Identitätsschlüssel)
- $EK_A$  (Alices Einmalschlüssel)
- $IK_B$  (Bobs Identitätsschlüssel)
- $SPK_B$  (Bobs hinterlegter, signierter Schlüssel)
- $OPK_B$  (Bobs hinterlegter Einmalschlüssel)

Die drei Diffi-Hellman Ausführungen sind:

1.  $IK_A$  und  $SPK_B$
2.  $EK_A$  und  $IK_B$
3.  $EK_A$  und  $SPK_B$

Die drei Ergebnisse werden aneinander gehangen und einer *Key-Derivation-Function (KDF)* übergeben. Das Ergebnis ist der geheime Schlüssel mit dem die Nachricht letztendlich ver-

schlüsselt wird.

Sollte optional ein Einmalschlüssel von Bob in dem Schlüsselset enthalten sein wird zusätzlich noch eine vierte Berechnung mit dem  $EK_A$  und  $OPK_B$  durchgeführt und zusammen mit den anderen der KDF übergeben. Die Funktionsweise des X3DH ist in Abbildung 1.1 zu sehen.

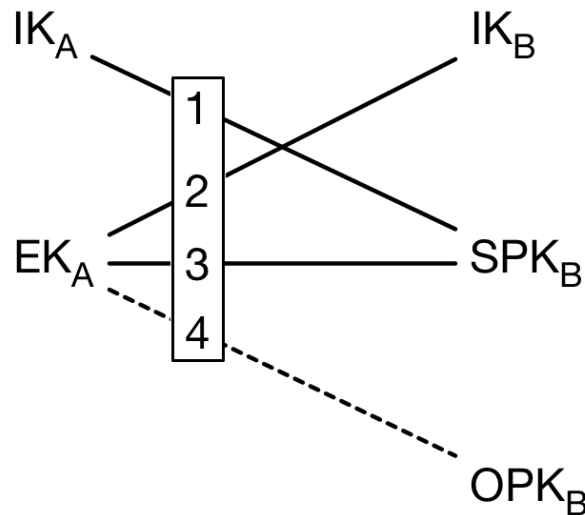


Abbildung 1.1: Visualisierung des X3DH [15]

Nach der Berechnung löscht Alice ihren privaten Einmalschlüssel und berechnet eine Byte-sequenz *Associated data (AD)* aus dem  $IK_A$  und  $IK_B$ , welche optional mit weiteren Informationen wie dem Accountnamen ergänzt wird. Anschließend kann Alice die erste Nachricht an Bob senden und damit das Gespräch beginnen.

Diese Nachricht enthält:

- $IK_A$
- $EK_A$
- welcher  $EK_B$  verwendet wurde
- die mit dem SK verschlüsselte Nachricht und dem AD als Anhang

Abhängig von den Sicherheitseinstellungen nutzt Alice den SK oder davon abgeleitete Schlüssel um Nachrichten an Bob zu senden.

## Empfang der ersten Nachricht

Analog zu den Berechnungen die Alice für den SK durchgeführt hat, verwendet Bob die entsprechenden privaten Schlüssel und Alices öffentliche Schlüssel. Durch den Zusammenhang des Public-Private-Key Verfahrens erhält Bob den gleichen SK und kann damit die empfangene Nachricht entschlüsseln.

### 1.3.2 Double Ratchet

Der Double Ratchet Algorithmus wird verwendet um Nachrichten verschlüsselt zwischen zwei Teilnehmer\*innen auszutauschen [16]. Dabei werden die Schlüssel aus den vorangegangenen Schlüsseln abgeleitet, so dass im Falle eines Leaks nur die Nachricht entschlüsselt werden kann, welche zu jenem geleakten Schlüssel gehört. Daraus können keine früheren oder nachfolgenden Nachrichten entschlüsselt oder die entsprechenden Schlüssel berechnet werden.

Das Herzstück des Algorithmus ist die KDF. Dabei wird eine KDF mit einem Schlüssel und Eingabedaten initialisiert und einer der Ausgabeschlüssel wird als KDF Schlüssel für die nächste KDF verwendet. So lässt sich von der Ausgabe nicht auf frühere Schlüssel schließen, selbst wenn die Eingabe bekannt ist. Bei genügend Entropie in den Inputs lässt sich auch nicht auf zukünftige Schlüssel schließen. Die Visualisierung der dadurch entstehenden KDF Kette lässt sich in Abbildung 1.2 nachvollziehen.

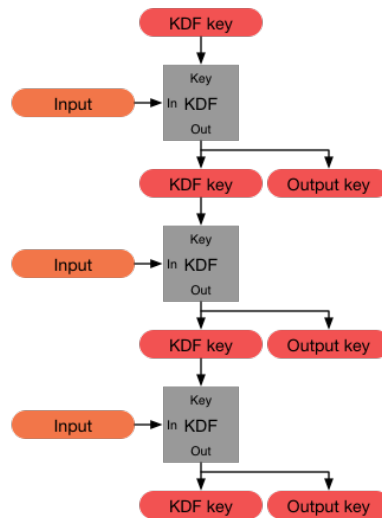


Abbildung 1.2: Visualisierung der KDF Kette [17]

Bei dem Double Ratchet Algorithmus gibt es drei Ketten:

- Stammkette
- Sendekette
- Empfangskette

Bei jeder Nachricht werden neue Schlüssel ausgetauscht, womit der neu berechnete SK als Schlüssel für die Stammkette verwendet wird. Dessen Ausgabe wird wiederum als Schlüssel für die Sende- und Empfangskette verwendet. Dieser Ablauf in der Stammkette wird als Diffi-Hellman Ratchet bezeichnet. Das Verfahren wird in Abbildung 1.3 dargestellt und verdeutlicht, wie mit dem zuletzt empfangenen öffentlichen Schlüssel (Bob) und dem neu generierten privaten Schlüssel (Alice) über das Diffi-Hellman Verfahren ein Schlüssel erzeugt wird. Bei dem Versand der Nachricht von Alice zu Bob wird der private Schlüssel des neu generierten Paares übermittelt und so kann Bob den gleichen Schlüssel berechnen. Die ständige Weiterentwicklung der beiden Ketten und die anschließende Nachrichtenver- oder Nachrichtenentschlüsselung mit deren Ausgaben wird als Symmetric-key Ratchet bezeichnet.

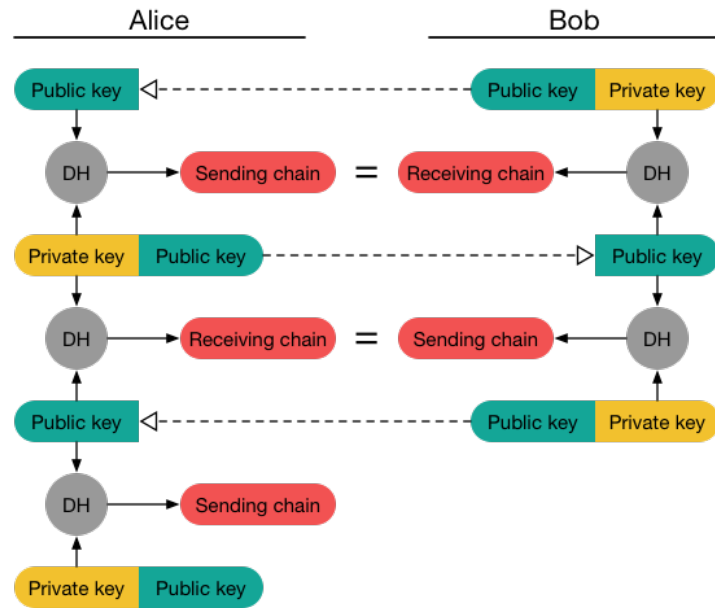


Abbildung 1.3: Visualisierung des Diffie-Hellman Ratchet Ablauf [18]

Der Double Ratchet ist die Hintereinanderschaltung des DH und des Symmetric-key Ratchets. Dabei wird vor jedem Nachrichtenversand ein neues Schlüsselpaar erzeugt und aus dem zuletzt erhaltenen öffentlichen Schlüssel und dem frisch erzeugten privaten Schlüssel mit dem Diffie-Hellman Ratchet ein neuer Schlüssel erzeugt, welcher anschließend für die Symmetric-key Ratchet verwendet wird. Bei jedem Nachrichtenversand wird der aktuelle eigene öffentliche DH-Schlüssel mitgeschickt. Der Zusammenhang zwischen den den Ratchets wird in Abbildung 1.4 verdeutlicht. Darin ist zu sehen, wie die mit dem DH-Ratchet generierten Schlüssel der Wurzelkette übergeben und deren Ergebnis für die Symetric-key Ratchet in der Sende-/Empfangskette weiterverarbeitet wird.

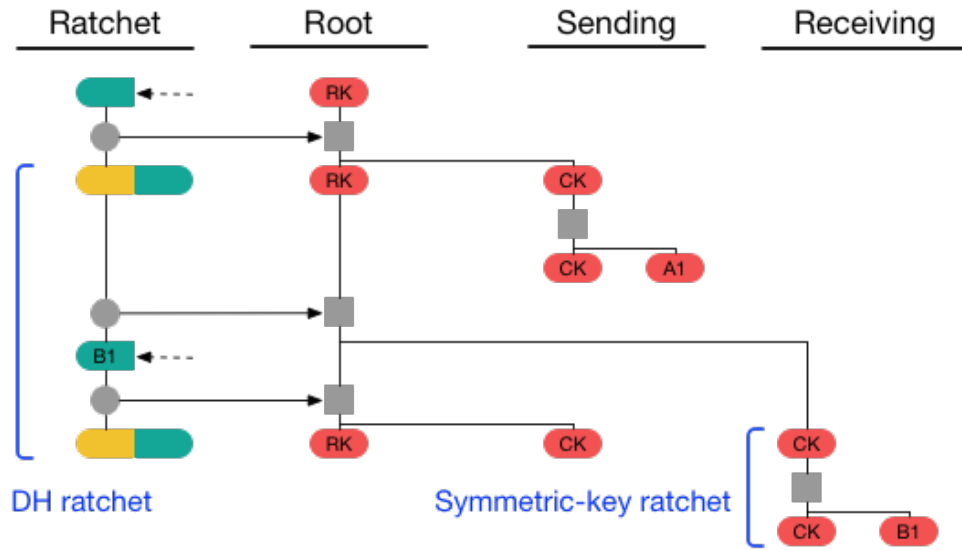


Abbildung 1.4: Visualisierung des Double Ratchet Ablauf [19]

Werden mehrere Nachrichten hintereinander versendet, ohne dass eine Nachricht empfangen wurde, wird, wie in Abbildung 1.5 dargestellt, die Ausgabe des letzten Symmetric-key Ratchet Durchlaufs als Eingabeschlüssel für den Nächsten verwendet. So ändern sich die Schlüssel für jede gesendete Nachricht. Da die Nachrichten auf der Empfangsseite analog durch die Empfängerkette laufen können diese entschlüsselt werden.

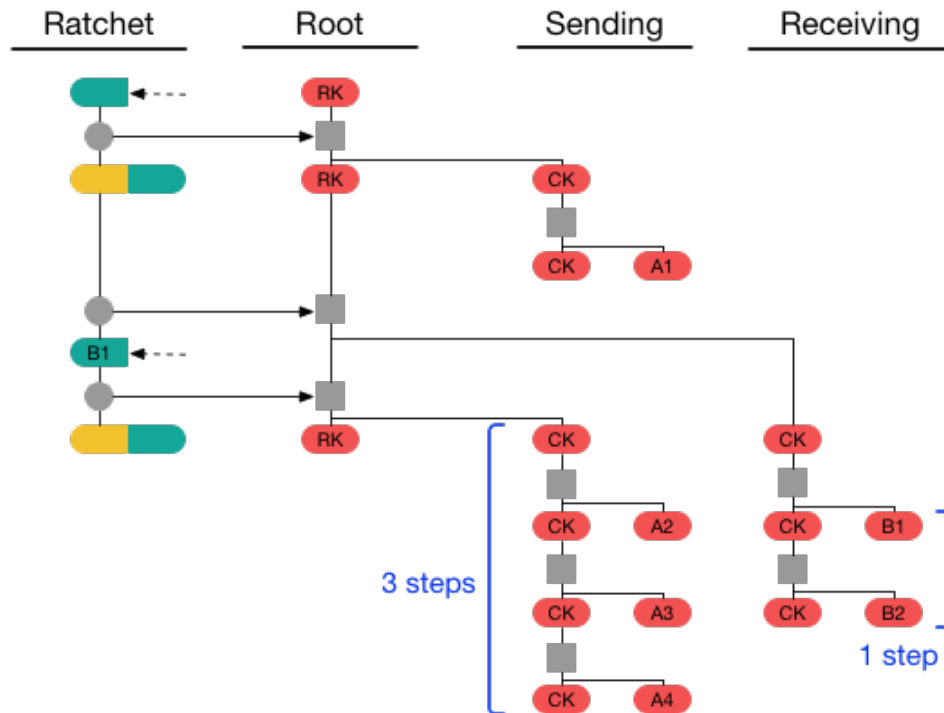


Abbildung 1.5: Visualisierung des Double Ratchet Ablauf Mehrfachversand [20]

Da so eine starke Abhängigkeit zur richtigen Verarbeitungsreihenfolge entsteht wird bei jeder Nachricht angegeben wie viele Nachrichten in der vorigen Sendekette waren und welche Position die Nachricht in der Aktuellen hat. Dies sorgt dafür, dass eine Anzahl vergangener Nachrichten auf dem Endgerät gespeichert werden, für den Fall dass einige in der falschen Reihenfolge empfangen werden.

### 1.3.3 XEdDSA und VEdDSA

Mit XEdDSA beziehungsweise VEdDSA [21] sind die Diffi-Hellman Funktionen für elliptische Kurven X25519 und X448 gemeint. VEdDSA ist eine Erweiterung von XEdDSA und sorgt dafür, dass es eine verifizierbar zufällige Funktion ist. Mit diesen Funktionen können Bytesequenzen signiert werden.

Mit diesen digitalen Signaturen wird sichergestellt, dass die Absender jene sind, für die sie sich ausgeben. Dafür wird das Public-Private-Key Verfahren angewendet. In diesem

Anwendungsfall werden die Nachrichten mit dem privaten Schlüssel des Absenders verschlüsselt und beim Empfänger mit dem öffentlichen Schlüssel des Absenders entschlüsselt. So wird der Absender verifiziert, da nur der Absender Zugriff auf seinen privaten Schlüssel hat.

### 1.3.4 Sesame

Mit Hilfe von asynchronen Schlüsselaustauschverfahren wie X3DH aus Abschnitt 1.3.1 können verschlüsselte Nachrichten versendet werden, auch wenn die Person mit der eine Unterhaltung begonnen werden soll offline ist. Zudem wird der Schlüssel mit dem Double Ratchet Algorithmus, wie in Abschnitt 1.3.2 beschrieben, nach jeder Nachricht verändert. Damit dieser ständige Wechsel von Schlüsseln, aber auch das Hinzufügen und Entfernen von Geräten zu Accounts, sowie das Einspielen von Backups funktioniert wird der Algorithmus *Session Management for Asynchronous Message Encryption (Sesame)* verwendet. Dieser regelt die Sitzungen, indem es immer eine aktive Sitzung pro Gerät gibt mit dem eine Kommunikation stattfindet und bei Bedarf Neue erstellt oder Alte gelöscht werden. Sollte eine Nachricht in einer inaktiven Sitzung empfangen werden, so wird sie zur aktiven Sitzung. Dadurch wird sichergestellt, dass die aktive Sitzung immer jene ist, mit der aktuell kommuniziert wird.

### Nachrichtenversand

Die zu sendende Nachricht wird verschlüsselt und mit dem Empfangs-IK, sowie einer Liste der Geräte-IDs an den Server gesendet, wo Sie an das empfangende Gerät weiter geleitet wird. Dabei werden sämtliche relevanten User-IDs mit aktiver Sitzung adressiert. Sollte der Server die Nachricht ablehnen wird das sendende Gerät informiert und bekommt gegebenenfalls die aktuellen Daten zugeschickt, um die Nachricht erneut zu verschicken.

### Nachrichtenempfang

Beim Empfang wird vom Server die verschlüsselte Nachricht, sowie User- und GeräteID gesendet. Falls bisher noch keine Kommunikation mit dem Sender stattgefunden hat, werden die öffentlichen Schlüssel aus dem Nachrichten-Header extrahiert und eine neue Sitzung mit den empfangenen Daten erstellt. Sollte bereits eine inaktive Sitzung existieren wird sie reaktiviert.



## 1.4 Signal-CLI

Für die Kommunikation mit den Signal Servern wird die *signal-cli* [22] verwendet, da sie einfach zu benutzen, stabil und quelloffen ist. Das erste Release wurde im Juli 2015 veröffentlicht und wird seitdem stetig weiterentwickelt. Es ist in Java geschrieben, weshalb das Projekt, wie in Abschnitt 1.4.3 beschrieben, dauerhaft als Daemon läuft und so nicht pro Nachricht/Nutzung die *Java Runtime Environment (JRE)* neu gestartet werden muss.

### 1.4.1 Installation

Dieser Abschnitt beschreibt die Installation der *signal-cli* auf einem Raspberry Pi 4. Hierbei ist zu beachten, dass frühere Hardwareversionen aufgrund der eingebauten Prozessorgeneration nicht unterstützt werden. Es sei zu beachten, dass JRE in der Version 17 oder höher installiert wird. Gegebenenfalls ist dafür eine aktuelle Linux-Version vonnöten.

#### Installationsvorbereitung

Für die Installation und spätere Ausführung sind verschiedene Programme notwendig. Im Falle der Installation auf dem Raspberry Pi müssen die *signal-cli*, sowie einzelne Bibliotheken manuell kompiliert werden, da keine kompilierte Variante bereitsteht.

- Installation der JRE  
`$ sudo apt-get install default-jre`
- Installation von cmake, clang und libclang:  
`$ apt-get install clang libclang-dev cmake make`
- Installation von Gradle  
`$ sudo apt-get install gradle`

- Installation von Rust

Bei der Installation von Rust auf einem Raspberry Pi wird die Installation von *rustup - The Rust Language installer* von snapcraft [23] empfohlen.

- Installation des Snap Store  
`$ sudo apt install snapd`
- Neustarten des Pi  
`$ sudo reboot`

- Installation von core snap

```
$ sudo snap install core
```
- Installation von rustup

```
$ sudo snap install rustup --classic
```
- Installation des protocol buffer compiler

```
$ sudo apt install protobuf-compiler
```

### Kompilierung der signal-cli Bibliotheken

Die *signal-cli* benötigt die Bibliotheken *Libsignal-client* [24] und *Libzkgroup* [25]. Diese werden von GitHub heruntergeladen und anschließend kompiliert.

- Installation der Libsignal-client Bibliothek
  1. In das java Verzeichnis wechseln

```
$ cd java
```
  2. Kompilierung der Android Lib mit # auskommentieren
  3. 

```
$ sed -i "s/, ':android'//" settings.gradle
```
  4. 

```
$ ./build_jni.sh desktop
```

Die kompilierte Bibliothek befindet sich dann im Verzeichnis `libsignal-client` unter `target/release/libsignal_jni.so`

### Kompilierung und Installation der signal-cli

Dieser Abschnitt entspricht der signal-cli Anleitung zum Kompilieren [26].

1. Herunterladen des Git-Repositorys

```
$ git clone https://github.com/AsamK/signal-cli.git
```
2. mit Gradle kompilieren

```
$ ./gradlew build
```
3. Einen shell wrapper im Verzeichnis `build/install/signal-cli/bin` erstellen

```
$ ./gradlew installDist
```
4. tar Datei im Verzeichnis `build/distributions` erzeugen

```
$ ./gradlew distTar
```

Die in Abschnitt 1.4.1 kompilierten Bibliotheken müssen in der kompilierten signal-cli tar ausgetauscht werden.

1. in das Verzeichnis `signal-cli/build/distributions` wechseln
2. tar entpacken

```
$ sudo ln -sf /opt/signal-cli-"${VERSION}"/bin/signal-cli /usr/local/bin/
```
3. libsignal-client aus der jar in `/opt/signal-cli-*/lib` löschen

```
$ zip -d signal-client-java-*.jar libsignal_jni.so
```
4. Die Bibliothek libzkgroup aus der jar löschen

```
$ zip -d zkgroup-java-*.jar libzkgroup.so
```
5. Pfad zu den kompilierten Bibliotheken in `/opt/signal-cli-*/bin/signal-cli` angeben

```
JAVA_LIBRARY_PATH="-Djava.library.path=/your/java/library/path"
exec "$JAVACMD" "$JAVA_LIBRARY_PATH" "$@"
```

### 1.4.2 Accounterstellung

Für die Erstellung eines Accounts wird eine Telefonnummer benötigt. Dabei ist der Empfang des Eröffnungscodes per SMS oder Sprachanruf möglich. In dieser Anleitung wird für den Account eine Festnetznummer verwendet und damit die Aktivierung per Sprachanruf. Zu beachten ist, dass bei der Telefonnummer die Ländervorwahl anzugeben ist, diese lautet für Deutschland +49.

1. Registrieren der Telefonnummer. USERNAME ist dabei durch die Telefonnummer zu ersetzen
  - mit SMS Verifikation `signal-cli -u USERNAME register`
  - mit Sprachverifikation `signal-cli -u USERNAME register --voice`
2. Sollte die Registrierung mit der Fehlermeldung *Captcha invalid or required for verification* fehlschlagen, muss ein *completely automated public Turing test to tell computers and humans apart (CAPTCHA)* gelöst werden.
  - a) Aufruf der Signal CAPTCHA Website [27] in einem Browser
  - b) das CAPTCHA lösen

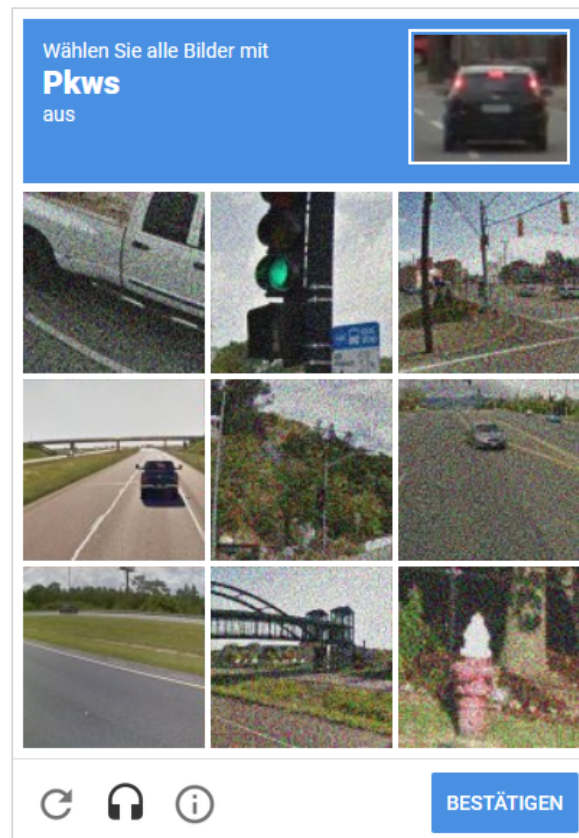


Abbildung 1.6: CAPTCHA Beispiel

- c) Über die Konsole der Entwicklungswerkzeuge kann das Token ausgelesen werden. In den meisten gängigen Browsern sind diese per F12 aufrufbar. Es ist zu beachten, dass die Website keinen weiteren Inhalt hat und somit gänzlich leer ist.



Abbildung 1.7: Token in der Firefox Konsole

Das Token ist hierbei der gesamte Inhalt nach der Redirect-Adresse

```
signalcaptcha://
```

- d) Nun wird der Registrierungsbefehl aus Punkt 1, mit dem zusätzlichen Parameter `--captcha 03AGdB...._9w`, erneut gestartet

3. Der per SMS oder Anruf erhaltene Code wird anstelle von `CODE` übergeben

```
signal-cli -u USERNAME verify CODE
```

### 1.4.3 Deamon

Bei der Einrichtung des Deamon Dienstes dient ein Artikel der c't 14/2021 [28] als Grundlage, welcher wiederum auf der Systembus-Anleitung der signal-cli aufbaut [29]. Dabei wird die signal-cli als Systemd-Dienst eingerichtet, so dass Sie beim Systemstart automatisch gestartet wird. Zudem wird der Dienst im Fehlerfall automatisch neu gestartet.

Wie in der c't und dem signal-cli Wiki beschrieben wird ein eigener User für die signal-cli eingerichtet. Dies ist eine übliche Praxis, um beispielsweise Zugriffsrechte einfacher gestalten zu können. Zudem haben im Anschluss alle User des Systems Zugriff auf die laufende signal-cli-Session über den D-Bus.

Der User `signal-cli` für den Deamon wird über den Befehl `sudo adduser --system --home /var/lib/signal-cli signal-cli` eingerichtet. Als root-User werden die Standard Konfig- und Servicedateien aus dem signal-cli Verzeichnis mit dem Befehlen `sudo cp data/org.asamk.Signal.conf /etc/dbus-1/system.d/` und `sudo cp data/org.asamk.Signal.service /usr/share/dbus-1/system-services/` kopiert.

```
1 [Unit]
2 Description=Send secure messages to Signal clients
3 Requires=dbus.socket
4 After=dbus.socket
5 Wants=network-online.target
6 After=network-online.target
7
8
9 [Service]
10 Type=dbus
11 Environment="SIGNAL_CLI_OPTS=-Xms2m"
12 ExecStart=/usr/local/bin/signal-cli --config /var/lib/signal-cli -u
    USERNAME daemon -system
13 User=signal-cli
14 BusName=org.asamk.Signal
15 # JVM always exits with 143 in reaction to SIGTERM signal
16 SuccessExitStatus=143
17 Restart=always
18
19 [Install]
20 Alias=dbus-org.asamk.Signal.service
```

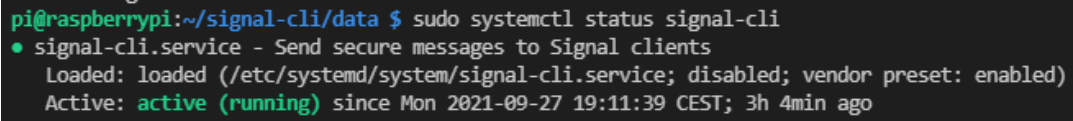
Abbildung 1.8: signal-cli.service

Die in Abbildung 1.8 dargestellte Service-Datei wird unter `/etc/systemd/system/signal-cli.service` abgelegt. Um die Datei dort zu speichern sind ebenfalls root-Rechte notwendig. Mit dem Eintrag `ExecStart=/usr/local/bin/signal-cli --config /var/lib/signal-cli -u USERNAME daemon --system` wird der Befehl für den Programmaufruf angegeben. Hier ist `USERNAME` wieder wie in Abschnitt 1.4.2 zu ersetzen.

In das dem Home-Verzeichnis des Users `signal-cli` werden die in Abschnitt 1.4.2 angelegten Kontodaten gespeichert. Über den Befehl `sudo cp -r ~/.local/share/signal-cli* /var/lib/signal-cli` werden sämtliche Dateien aus dem persönlichen Verzeichnis kopiert. Abschließend wird mit `sudo chown -R signal-cli /var/lib/signal-cli` eingestellt, dass sämtliche Dateien im angelegten Home-Verzeichnis dem gleichnamigen User gehören.

Der Service wird mit dem Befehl `sudo systemctl --now enable signal-cli` aktiviert und über `sudo systemctl start signal-cli` gestartet.

Mit dem Befehl `sudo systemctl status signal-cli.service` kann im Anschluss kontrolliert werden, ob die Einrichtung erfolgreich war und ob der Service aktiv ist.



```
pi@raspberrypi:~/signal-cli/data $ sudo systemctl status signal-cli
● signal-cli.service - Send secure messages to Signal clients
   Loaded: loaded (/etc/systemd/system/signal-cli.service; disabled; vendor preset: enabled)
   Active: active (running) since Mon 2021-09-27 19:11:39 CEST; 3h 4min ago
```

Abbildung 1.9: Status signal-cli.service

Abschließend kann über die Kommandozeile mit `signal-cli --dbus-system send -m "Hello , World!" RECEIVER` eine Testnachricht an einen anderen Signal-Account geschickt werden. Für RECEIVER muss die Telefonnummer des Signal-Accounts inklusive Ländervorwahl angegeben werden. Es ist nicht notwendig den User mit dem Parameter `-u USERNAME` anzugeben, da dieser bereits in der Service-Datei hinterlegt ist.

## 2 Funktionalitäten und Module

In diesem Kapitel werden die einzelnen Funktionalitäten und Module des Bots beschrieben. An einigen interessanten Stellen wird auf die Herangehensweise der Programmierung eingegangen. Da sich der Bot in einem frühen Stadium befindet und primär als Basis für zukünftige Projekte dienen soll, sind die Funktionalitäten breit gefächert. Es soll erforscht werden, welche Möglichkeiten dem Bot zur Verfügung stehen, um Folgeprojekte gezielter und komplexer auszubauen.

In diesem Projekt wurde ein agiler Programmieransatz gewählt. Im Wochentakt wurden die einzelnen Funktionalitäten implementiert und von verschiedenen Anwender\*innen und Gruppen im Dauerbetrieb getestet. Dabei wurde stets darauf geachtet, wie oft die neuen Befehle des Bots genutzt wurden, ob es zu Fehlern kommt und welche Verbesserungsmaßnahmen getroffen werden können.

### 2.1 Echos und Statische Texte

Nach der Installation der Signal CLI aus Abschnitt 1.4 sollte die einfachste Art Bot-Antwort getestet werden. Ein beliebter Test für Kommunikationsanwendung stellt die echo-Funktion dar. Dabei bekommt die sendende Person oder Maschine den gleichen Nachrichteninhalt zurück, den sie abgeschickt hat. So kann ein *Proof of Concept (PoC)* realisiert werden. In diesem Projekt wurde ein ähnlicher Ansatz gewählt: der Bot sollte von einem User privat angeschrieben werden können oder die Nachrichten einer Test-Gruppe auslesen, um darauf zu reagieren. Sobald der Befehl `.echo` geschickt wurde, sollte er ebenfalls mit *echo* antworten. Dabei ist das erste Grundprinzip für den Bot entstanden. Alle Nachrichten, die nicht mit exakt einem Punkt anfangen, sollen sofort verworfen werden, alle anderen werden auf ein Kommando ausgewertet. So muss nicht jede einzelne Nachricht analysiert werden und die Nutzer\*innen haben nicht das Gefühl, dass der gesamte Schriftverkehr dokumentiert wird.

Auf dieser Basis wurden eine Menge einfacher Kommandos angelegt, die einen statischen String zurückgeben. Um den Code gering zu halten wurden diese in einer Datei im *JavaScript Object Notation (JSON)* Format ausgelagert, welche beim Start als Datentyp *Dictionary (Dict)* eingelesen wird. So kann bei einer Nachricht mit Punkt der Inhalt



bis zur ersten Leertaste als Kommando interpretiert und als Schlüssel im sogenannten `FUN_DICT` gesucht werden, falls vorher kein anderes, komplexeres Kommando erkannt wurde. Falls die Nachricht mit einem Punkt beginnt, aber das Kommando nicht gefunden wurde, gibt der Bot eine Rückmeldung wie in Abbildung 2.1 zu sehen ist.

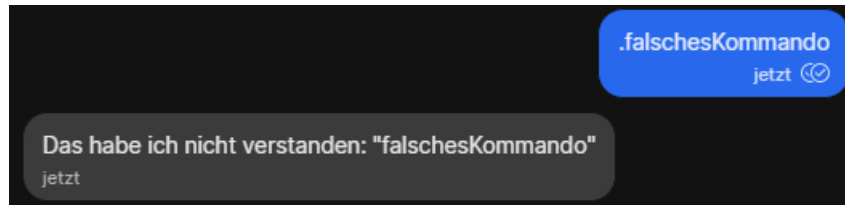


Abbildung 2.1: Statische Texte im Signalbot

Um einen Überblick über alle verfügbaren komplexeren Funktionen zu behalten wurde das, bereits in Abschnitt 1.2 erwähnte, `.help` Kommando implementiert. Dieses gibt eine Liste aller verfügbaren Kommandos zurück. Sobald mal mit `.help <commandname>` ein vorhandenes Kommando der `help`-Funktion als Parameter übergibt, meldet der Bot eine genauere Auskunft über den Befehl zurück.

Zuletzt werden alle unbekannten Kommandos und Fehler durch einen Logger dokumentiert. Da der Bot permanent laufen soll und der Dienst sich bei einem schwerwiegendem Fehler neu startet kann so analysiert werden, welche Anomalien behoben werden müssen.

## 2.2 Programmierschnittstellen

Eine *Application Programming Interface - Programmierschnittstelle (API)* ist ein offengelegter Teil eines Softwaresystems, der es einem anderen Programm ermöglicht bestimmte Ressourcen abzufragen. Die Nutzung verschiedener APIs sind für Bots äußerst wertvoll, da man mit wenigen Zeilen Code viel Funktionalität anbieten kann. Zudem wurde in diesem Projekt eine frühe Implementierung der APIs bevorzugt, um zu beobachten ob der Bot auf Langzeit ohne Einschränkungen mit webbasierten Diensten kommunizieren kann. Bis dato wurde die Dadjoke-API [30], die Foolproof Decision-Making API [31], sowie die DeepL API [32] implementiert. Bei allen APIs ist die Arbeitsweise vergleichbar. Bei den Befehlen `.dadjoke`, `.yesno` oder `.yn` oder `.deepl` wird eine *Representational State Transfer (REST)* Get-Request an den jeweiligen API-Server gestellt. Dafür wird die Requests-Bibliothek [33] genutzt. Die Antwort ist ein String im JSON-Format, wel-

cher mithilfe der gleichnamigen Bibliothek in den Datentyp Dict übersetzt werden kann. Dictionaries sind syntaktisch ähnlich zu JSON und bieten eine Möglichkeit an, um mit Schlüssel-Wert-Paaren zu arbeiten. Auf Basis dieses Dicts kann die Antwort des Bots in Form von Text und optionalem grafischem Anhang zusammengebaut und versendet werden. Da die Inhalte der Antworten bei jeder Anfrage individuell sind, muss die API bei jedem Bot-Command aufgerufen werden. In der Abbildung 2.2 wird die Funktionsweise des Commands `.yesno` dargestellt.

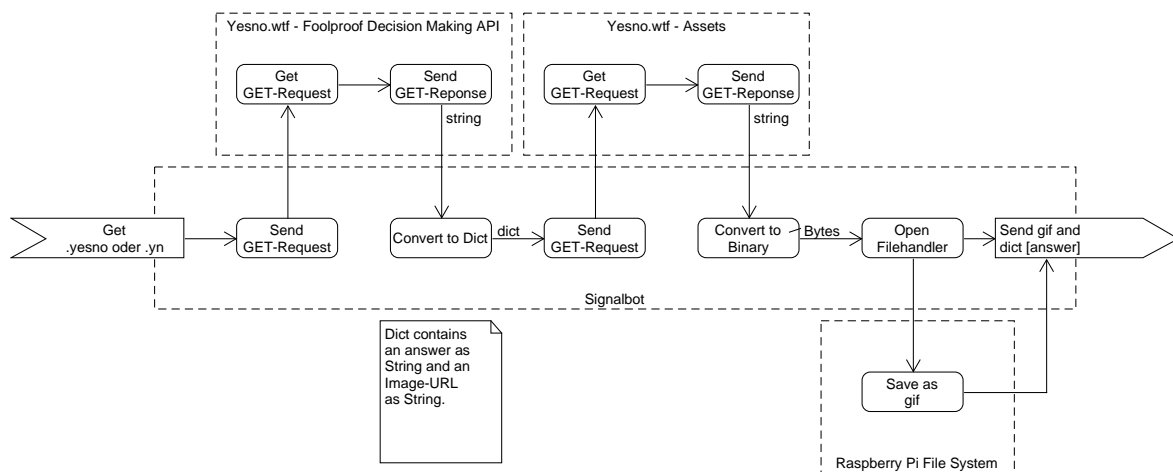


Abbildung 2.2: Funktionsweise des `.yesno`-Kommando

## 2.3 WebCrawler

WebCrawler stellen automatisierte Software dar, die eine bestimmte Anzahl von Webseiten analysieren, um gezielt Daten zu gewinnen. WebCrawler werden meist dann verwendet, wenn keine Programmierschnittstelle zur Verfügung gestellt wird und wenn es möglich ist Webseiten durch Regelmäßigkeiten auszulesen. Zwei Beispiele stellen hier die Seiten `freegamesyo.com` und `geschichtgndern.de` dar. Beide sind in einer regelmäßigen Struktur aufgebaut und werden häufig von den Personen genutzt, für die der Signalbot geschaffen wurde.

Für das Crawling muss ein String im *Hypertext Markup Language (HTML)*-Format abgefragt und ausgewertet werden. Ersteres wird äquivalent zu Abschnitt 2.2 mithilfe einem einfachen Get-Request erledigt. Für die Auswertung wird die Bibliothek *Beautiful Soup* [34] genutzt. Das Crawling von `freegamesyo` ist vergleichsweise simpel: Alle kostenlosen

Spiele liegen in Form des Titels, Links und Bildes in einer ungeordneten Liste vor. Es muss also nur über alle 11-Elemente der Webseite iteriert werden. Dabei stellt jeder Iterationsschritt ein Spiel dar. Der Bot speichert das Bild des aktuellen Spieles temporär ab und schickt dann eine Nachricht samt Anhang pro Spiel an die anfragende Person oder Gruppe zurück.

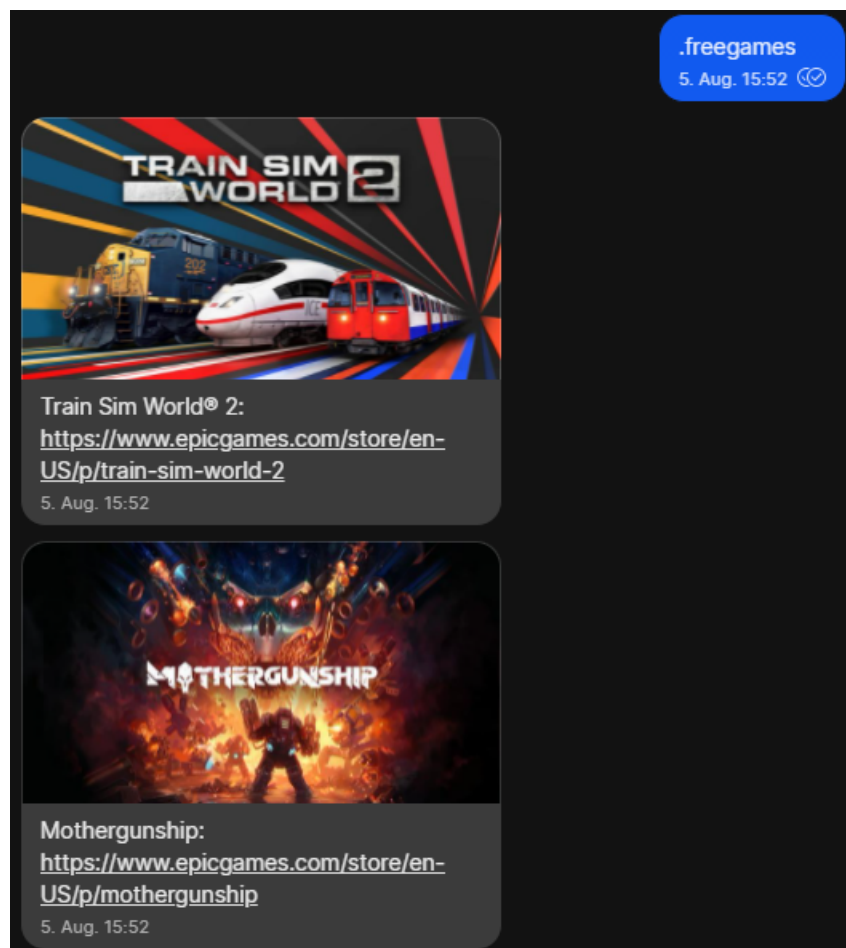


Abbildung 2.3: Antwort des Bots beim `.freegames`-Kommando

Mit der Einführung der zeitgesteuerten Events aus Abschnitt 2.4 wurde eine Abonnentenliste für das `.freegames`-Kommando eingeführt. Dabei crawlt der Bot in regelmäßigen Zeitabständen `freegamesyo`. Bei einer Veränderung der Spieleliste im Vergleich zum letzten Crawling wird allen abonnierenden Personen und Gruppen die neue Liste geschickt. Unabhängig von der Entscheidung für oder gegen das Abonnement kann wie gewohnt der Befehl `.freegames` genutzt werden.

Das Kommando `.gn` oder `.geschicktgendern` ist eines der komplexesten Features des Bots, da es auf der Webseite diverse Ausnahmefälle gibt, welche abgefangen werden müssen. Zudem soll der Bot ganze deutsche Sätze oder sogar Texte auf Genderneutralität prüfen. Dafür muss zuerst eine verlässliche Datenbasis aus der Webseite gebaut werden.

---

**Algorithm 1:** Update der genderneutralen Datenbasis

---

**Input:** Geschicktgendern.de per REST-GET-Request

**Output:** genderneutralDictionary as gn

```

1 value, key, keyOriginal  $\leftarrow$  " " ;
2 Get all tds;
3 for td in tds do
4     key, keyOriginal  $\leftarrow$  td.text;
5     if key = value  $\vee$  key[0:3] = "... " then
6         continue
7     end if
8     Get next Sibling from td;
9     if sibling then
10        value  $\leftarrow$  sibling.text;
11    else
12        value  $\leftarrow$  "NONE" ;
13        continue
14    end if
15    if  $\neg$ value or "noch kein passender Begriff gefunden;" in value then
16        continue
17    end if
18    if ["(", "[", "{", ")", "]", "}"] in key then
19        Remove all brackets and their content;
20    end if
21    if key in gn then
22        gn[key][keyOriginal]  $\leftarrow$  value;
23    else
24        gn[key]  $\leftarrow$  {keyOriginal = value};
25    end if
26 end for
27 return gn

```

---

Wie in dem Algorithmus 1 zu sehen ist liegen die benötigten Informationen von geschickt-

gendern.de in einer Tabelle vor. Es kann also über alle *table data (td)*-Elemente der Seite iteriert werden, wobei das erste Element der Schlüssel des dicts wird, während der Inhalt des Geschwisterelementes den Wert bildet. Dabei werden alle Ausnahmen raus gefiltert, die nicht für Datenbasis des Bots relevant sind. Dazu gehören Einträge die leer sind, noch keinen passenden Begriff beinhalten, kein Geschwisterelement besitzen und Beispielsätze die mit drei Punkten beginnen. Danach wird der Schlüssel so weit wie möglich vereinfacht. Es gibt Wörter, die kontextbezogen eine unterschiedliche Bedeutung haben oder im Singular sowie im Plural gleich geschrieben werden. Da der Bot keine Möglichkeit zum Überprüfen von Kontexten besitzt müssen alle verfügbaren Alternativen zu einem Wort angezeigt werden. Dafür wird ein Key erzeugt, welcher nur das einzelne Wort ohne Zusätze jeglicher Art enthält. Der Wert von Key ist dann ein Objekt mit feingranularer Aufteilung aus der dann die Antwort des Bots erzeugt wird. Ein Beispiel lässt sich der Abbildung 2.4 entnehmen.

```
"lehrer": {  
  "Lehrer (sg.)": "Lehrperson; Lehrkraft",  
  "Lehrer (pl.)": "Lehrkräfte; Lehrpersonen; das Kollegium; Lehrende; Unterrichtende"  
},  
  
ntp.SignalBot  
Potentiell "lehrer" gefunden. Alternativen:  
  Lehrer (sg.): Lehrperson; Lehrkraft  
  Lehrer (pl.): Lehrkräfte; Lehrpersonen; das Kollegium; Lehrende;  
  Unterrichtende  
jetzt
```

Abbildung 2.4: Auszug aus der gn.json und resultierende Antwort des Bots beim .gn-Kommando

Bei der Nutzung des .gn-Befehls können einzelne Wörter aber auch mehrere Sätze übergeben werden. Dafür wird der Übergabestring von allen Sonderzeichen bereinigt und in eine Liste einzelner Wörter aufgeteilt. Die Ausgabe aus Abbildung 2.4 wird dann für jedes gefundene Wort verlängert. Da die Webseite geschicktgendern.de über 1.800 Einträge enthält, kann sie nicht bei jedem Aufruf der .gn-Funktion neu gecrawlt werden. Ein häufiges Crawlen bedeutet nicht nur einen hohen Leistungsaufwand und eine hohe Reaktionszeit des Bots, es kann auch passieren, dass die IP des Bots von der Webseite blockiert wird. Letzteres ist tatsächlich bereits passiert, als nur beim Start des Bots die Datenbasis aktualisiert wurde, aber der Bot aufgrund einer Fehlersuche häufig neu gestartet werden musste. Eine Abhilfe schaffen hier die zeitgesteuerten Events aus Abschnitt 2.4. In der jetzigen Version wird die gn.json einmal in der Woche aktualisiert.

## 2.4 Zeitgesteuerte Events

Einer der Kernanforderungen des Bots war es einzelnen Personen einer Gruppe zum Geburtstag zu gratulieren. Dafür werden zwei Funktionalitäten benötigt. Zum einen müssen alle Personen und die dazugehörigen Geburtstage abgespeichert werden. Dies wird im Abschnitt 2.5 näher erklärt. Zum anderen muss täglich geprüft werden, ob einer oder mehrere der abgespeicherten Geburtstage auf das jeweilige Datum fallen. Dafür wurde eine Möglichkeit implementiert, um zeitgesteuerte Events in Python zu nutzen. Einen Hauptteil der Arbeit nimmt die Bibliothek *Schedule* [35] ab. Beispielsweise lässt sich mit der Zeile `schedule.every().hour.do(job)` jede Stunde die Funktion `job()` aufrufen. Dabei ist zu beachten, dass die Bibliothek *Schedule* folgende Dinge nicht mit sich bringt:

- **Jobpersistenz**

Die Schedule-Bibliothek hat keine automatisierte Funktionalität, die abspeichert welche Jobs bereits durchgeführt wurden oder noch durchzuführen sind. Diese Informationen geht bei einem, meist ungewolltem, Neustart verloren. Hier ist also darauf zu achten, dass in der `init()`-Funktion alle wichtigen Jobs neu gestartet werden. Eine andere Form der Persistenz ist nicht gefordert.

- **Exakte Funktionsausführung**

Die Schedule-Bibliothek kann nur in der Auflösung von Sekunden Jobs durchführen. Eine Ausführung mit Präzision im Millisekundenbereich ist demnach nicht möglich. Allerdings wird diese auch nicht benötigt.

- **Lokalisation**

Es wird keine Unterstützung für unterschiedliche Zeitzonen, sowie Sommer- und Winterzeit angeboten. Demnach wird immer die Zeitzone des unterliegenden Betriebssystems gewählt. Da alle Nutzer\*innen des Signalbots in Deutschland leben stellt dies kein Problem dar.

- **Gleichzeitige Funktionsausführung**

Sobald mehrere Events zu einem gleichen Zeitpunkt ausgeführt werden kann es passieren, dass der Scheduler nicht alle Events ausführt. Dafür muss eine Funktionsverteilung in verschiedene Threads passieren. Dafür wurde eine Wrapper-Funktion `run_threaded()` eingebaut, mit der es möglich ist einen Job in einem eigenem Thread zeitgesteuert zu starten. Somit ist es möglich mehrere Events zur gleichen Zeit passieren zu lassen.

Neben den Events aus Abschnitt 2.5 werden noch 2 statische Funktionen für das GG

E-Sport Jugendzentrum programmiert, die den Arbeitsalltag erleichtern sollen. So erinnert der Signalbot die haupt- und ehrenamtlichen Mitarbeiter\*innen in regelmäßigen Zeitabständen daran sich in den Schichtplan einzutragen und die gearbeiteten Stunden zu dokumentieren.

## 2.5 Datenbanken

Einer der initialen Wünsche war, dass der Bot automatisch an den Geburtstagen der einzelnen Gruppenmitglieder gratuliert. Die Basis dafür wurde bereits im Abschnitt 2.4 geschaffen. Mit dem zeitgesteuerten Events kann automatisch einmal am Tag geprüft werden, ob eine Person Geburtstag hat. Nun musste nur noch eine überprüfbare Datenbasis und die Möglichkeit zur Beschreibung dieser geschaffen werden. Da die einzuspeichernden Geburtstage immer nach dem gleichen Schema, nämlich Name und Datum, abgespeichert sind, wurde für die Datenbasis das Format *Comma/Character-separated values (CSV)* gewählt. CSV ist im Gegensatz zu JSON ein starrereres, weniger vielseitiges Format ohne hierarchische Struktur. Dafür ist es kompakter, speicherärmer und es können ohne Probleme Zeilen mit dem Append-Modus des Filewriter-Moduls angehängt werden. Bei der Initialisierung des Bots werden alle bereits gespeicherten Geburtstage in einer Liste aus Geburtstags-Objekten abgespeichert. Der `.saveBirthday`-Befehl nimmt als Parameter einen Namen und ein Datum im Format DD.MM entgegen, versucht diese zu Parsen, erstellt ein Geburtstags-Objekt und fügt der CSV eine neue Zeile hinzu. Dabei wird auf Richtigkeit des Datums, sowie auf Duplikate in der Datenbank geprüft. Das Kommando `.allBirthdays` gibt alle eingespeicherten Geburtstage zurück. Zuletzt gibt es mit `.nextBirthday` die Möglichkeit den nächsten Geburtstag anzeigen zu lassen. Dafür wird die Zeit in Tages bis zum eingespeicherten Datum mit Beachtung eines möglichen Jahresüberlaufes berechnet. Die Ausgabe des Bots ist in Abbildung 2.5 zu sehen.

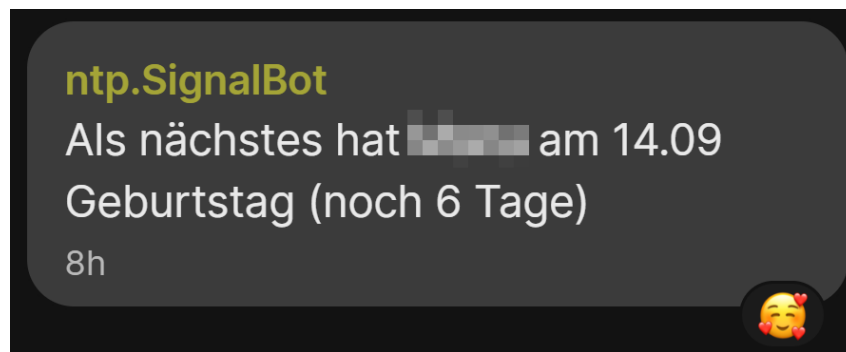


Abbildung 2.5: Antwort des Bots beim `.nextBirthday`-Kommando

Mit dem Berechtigungssystem aus Abschnitt 2.7.1 wurde die Möglichkeit geschaffen für jede Gruppe eine eigene CSV-Datei anzulegen. In Zukunft sollen diese in einer Datenbank auf Basis von SQL oder MongoDB vereint werden, um den Verwaltungsaufwand zu vereinfachen. Das Abspeichern von kommenden Events und Zitaten sind ähnlich zu den Geburtstagen aufgebaut und benötigen keine nähere Betrachtung.

## 2.6 M2M-Kommunikation

Unter *Machine to Machine (M2M)*-Kommunikation wird die automatische Kommunikation zwischen zwei Maschinen, meistens Computern, verstanden. Dies kann als 1 zu 1, 1 zu n oder auch n zu n Beziehung realisiert werden. Als Beispiel für eine M2M-Kommunikation wird eine *Message Queuing Telemetry Transport (MQTT)* Schnittstelle implementiert.

MQTT ist ein weit verbreitetes Protokoll [36], welches nicht für Echtzeitkommunikation, sondern für sporadische Statusmeldungen in Netzwerken geschaffen wurde bei denen der Datendurchsatz gering ist oder die Verbindung nicht konstant aufrechterhalten wird. Dadurch können stromsparende IoT-Lösungen realisiert werden, da keine konstante Internetverbindung notwendig ist.

Bei MQTT kommunizieren Clients nicht direkt miteinander, sondern senden Ihre Daten an einen Server (Broker) und bekommen Daten vom Broker zugeschickt. Das Senden von Daten an den Broker wird `publishen` genannt. Diese werden einem `topic` zugeordnet und darauf können sich andere MQTT Clients `subscribe`. Beim `subscribe` bekommen die Clients die Daten vom Broker zugesendet, welche unter dem `subscribten topic` gepublishet wurden. Dadurch kann die Kommunikation bidirektional oder unidirektional realisiert werden.



Im Signalbot wird ein Client mit der Bibliothek `paho.mqtt.client` [37], welche unter der EPL V1.0 Lizenz steht, erzeugt. Dieser Client läuft während der gesamten Laufzeit des Signalbots in einem eigenen Task.

```
1 def mqtt_client():
2     client2 = mqtt.Client()
3     client2.on_message = on_mqtt_message
4     client2.username_pw_set(MQTT_USER, MQTT_PWD)
5     client2.connect(MQTT_IP, MQTT_PORT, MQTT_TIMEOUT)
6     client2.subscribe("test/temperature")
7     client2.loop_forever()
```

Abbildung 2.6: Signalbot MQTT Client

Für die Initialisierung des Clients wird sowohl ein User, als auch ein Passwort für die Authentisierung benötigt, um den Datenverkehr abzusichern. Über die `subscribe()` Funktion können beliebige topics subscribed werden. Sobald in einem Daten gepublished werden, wird die Callback-Funktion `on_mqtt_message` aufgerufen und dort werden die Daten verarbeitet.

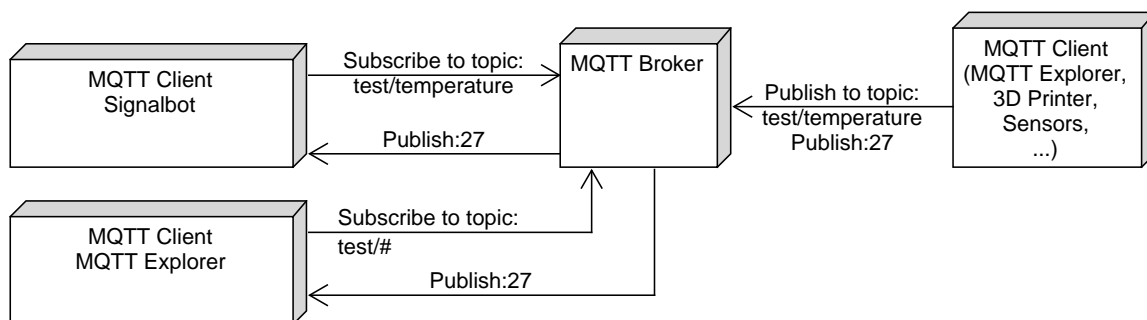


Abbildung 2.7: Eingesetzte MQTT Architektur

In der Grafik 2.7 ist die für dieses Projekt umgesetzte Architektur zu sehen, um den Anwendungsfall zu testen. Dabei wird im Signalbot ein MQTT client erstellt, welche das topic `test/temperature` subscribed. Wann immer Daten unter diesem topic gepublished werden, wird eine Nachricht mit dem Wert und topic an einen Signalaccount gesendet. Zur Kontrolle der Kommunikation werden mit dem Programm MQTT Explorer [38] sämtliche topics subscribed, welche mit `test/` beginnen. Dafür wird mit die Wildcard [39] `#` verwendet, welche ein Platzhalter für sämtliche topics und sub-topics ist die in diesem Fall mit

test/ beginnen.

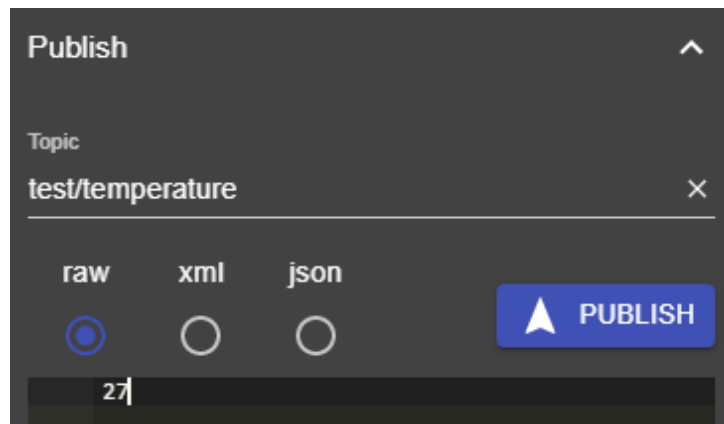


Abbildung 2.8: Senden von Testnachrichten via MQTT Explorer

Wird eine Nachricht als Test über den MQTT Explorer an das im Signalbot abonnierte topic gesendet 2.8, registriert der Signalbot dies innerhalb kürzester Zeit und leitet die Informationen über das Signal-Protokoll weiter an einen hinterlegten Account 2.9.

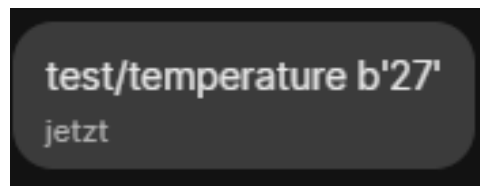


Abbildung 2.9: Empfang der MQTT Nachricht via Signal

Dies stellt einen PoC dar und kann wie in 3.2 erwähnt einen industriellen Einsatz bekommen. Dabei wäre die Benachrichtigung bei Events oder Grenzwertabweichungen per Signal, für eine schnelle Reaktion denkbar. Für dieses Projekt wurde der Status und die Temperatur eines 3D Druckers als Datenquelle 2.10 verwendet. Dadurch kann eine Benachrichtigung versendet werden, sobald ein Druck beendet wurde.

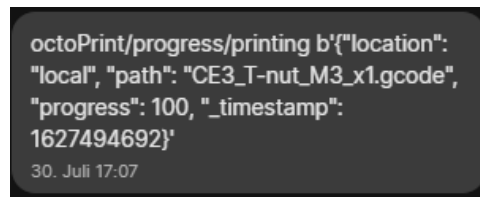


Abbildung 2.10: Empfang des 3D Drucker Status als MQTT Nachricht via Signal

## 2.7 Weitere Funktionalitäten

Dieser Abschnitt soll einige Features des Signalbots beleuchten, die in keine der obigen Abschnitte eingeordnet werden können.

### 2.7.1 Berechtigungssystem

Da nicht alle Gruppen alle Funktionalitäten benötigen und unerlaubte Nutzer\*innen und Gruppen nicht einfach den Bot nutzen sollen wurde in der *config.json* ein Berechtigungssystem angelegt. So kann für jeden einzelnen Befehl des Bots per Allowlist angegeben werden, wer berechtigt ist diesen zu nutzen. Die Kommandos *.help* und *.credits* sind immer verfügbar. Ersterer stellt dynamisch eine Liste der erlaubten Kommandos zusammen. Sollte ein Befehl genutzt werden, für den die Gruppe nicht zugelassen ist, wird eine Fehlermeldung ausgegeben. Im Anschluss wurde das Berechtigungssystem mit den Datenbanken aus Abschnitt 2.5 verbunden, sodass für jede erlaubte Gruppe eine eigene JSON zum jeweiligen Befehl angelegt wird. Auch das Kommando *.links* funktioniert über das Berechtigungssystem. Hierfür wird für jede Gruppe in der *config.json* ein Link-Objekt angelegt.

### 2.7.2 Formelparser

Im Projektzeitraum hat sich die testende Gruppe mit dem Viererproblem beschäftigt. Das Problem lässt sich wie folgt beschreiben: "Given no more than four instances of the digit 4, represent all integers using a finite number of mathematical symbols and operators in common use." [40] Da die Lösung des Problems zum größten Teil per Signal-Textnachricht stattgefunden hat wurde hier ein Feature gewünscht, welches eine getippte Formel in ein Bild umwandelt. Um diese zu parsen wird die Funktion *sympify()* der Bibliothek

sympy [41] verwendet. Die Darstellung als Bild erfolgt mittels einer Zweckentfremdung der matplotlib-Diagrammfunktion und ist in Abbildung 2.11 zu sehen.

$$\frac{\frac{x^2}{\sqrt{r^2-x^2}} + \sqrt{r^2-x^2}}{r^2-x^2} = -\frac{x^2}{\sqrt{r^2-x^2}(r^2-x^2)} - \frac{\sqrt{r^2-x^2}}{r^2-x^2}$$

$$-((\sqrt{r^2-x^2} + (x^2/(\sqrt{r^2-x^2}))))/(r^2-x^2) = -(\sqrt{r^2-x^2}/(r^2-x^2)) - ((x^2/(\sqrt{r^2-x^2}))/((r^2-x^2)))$$

jetzt

Abbildung 2.11: Antwort des Bots beim `.tex`-Kommando

Da das SymPy Development Team annimmt, dass alle Gleichungen so umgestellt sind, dass sie rechtsseitig den Wert 0 haben wurde der Parser leicht erweitert. Bei Benutzung des `.tex`-Kommandos wird eine Gleichung an jedem Gleichheitszeichen aufgeteilt und einzeln geparkt. Dann wird das Ergebnis wieder zusammengesetzt. So können auch Gleichungen mit einer beliebigen Anzahl von Gleichheitszeichen dargestellt werden. Zudem wurde der Befehl so programmiert, dass immer der optimale Platz des Anhangs genutzt wird. Dennoch befindet sich der `.tex`-Befehl noch in einem experimentellen Stadium. An manchen Stellen werden Formeln durch SymPy automatisch umgestellt, was die Darstellung von genauen Rechenwegen erschwert. Auch Formelzeichen wie  $\pm$  werden bis dato nicht korrekt geparkt.

### 2.7.3 Lockdownchallenge

Die Jugendkirche Düsseldorf hat Anfang 2021 eine Reihe von kleinen Herausforderungen entworfen, die dabei helfen sollen sich selbst aus dem Alltagstrott des Lockdowns zu heben. Bei diesem Projekt hat das Signalbot-Team ehrenamtlich mit Pädagog\*innen zusammengearbeitet. Das Endziel war es ein kleines Büchlein zu produzieren, in dem jeden Tag eine Seite des Buches eine neue Herausforderung stellt. Aus verschiedenen Gründen wurde diese Idee nicht umgesetzt. Allerdings wurde diese Basis genutzt um das Kommando `.challenge` zu entwerfen. Der Signalbot ist nun ein Teil der Jugendkirchengruppe. Bei Benutzung des Kommandos bekommt die Person eine zufällige Herausforderung und hat das Recht mit dem Abschluss der Aufgabe zu protzen.

## 2.8 Nicht umgesetzte Funktionalitäten

Ein anfänglicher Wunsch war die Implementierung einer Abstimmungsfunktion, welche dem cvh-bot ähnelt. Dabei sollte eine Abstimmung mit einem Timer starten können. Während des Zeitraumes soll der Bot alle Nachrichten zählen die den Inhalt ++,+,0/o,-,- enthalten. Falls eine Person zweimal abstimmt, wird die ältere Wahl verworfen. Am Ende der Abstimmung wäre mit der Matplotlib ein Auswertungsdiagramm entstanden. Allerdings hat Signal bereits eine einfache Art und Weise um Abstimmungen durchzuführen. Da Nutzer\*innen auf die Nachrichten anderer Personen mit genau einem Emoji reagieren können, kann so eine Abstimmung mit nur einer Textnachricht vollzogen werden. Durch die Anzahl der einzelnen Reaktionen lässt sich, wie in Abbildung 2.12 zu sehen, direkt ein Ergebnis ablesen. Dementsprechend hat sich die Implementierung einer eigenen Abstimmungsfunktion redundant angefühlt und wurde auch bisher nicht per `.request` angefragt.

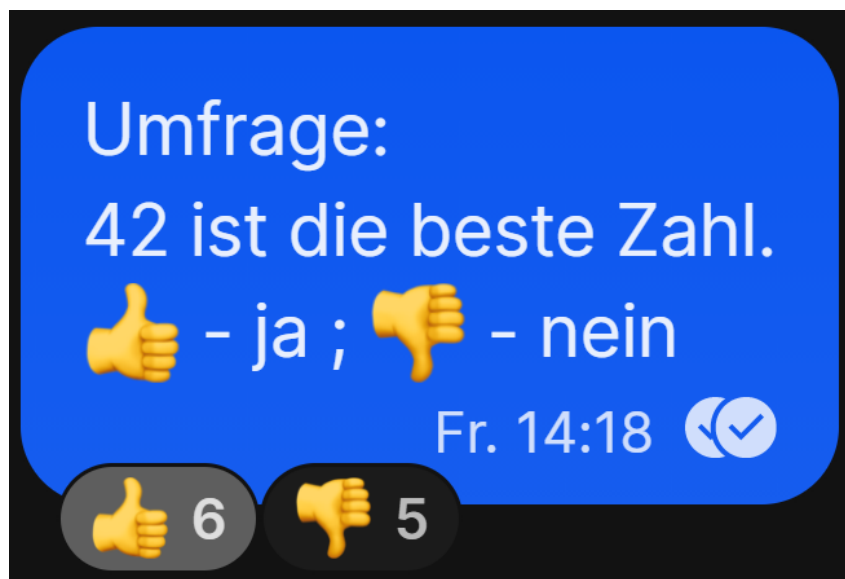


Abbildung 2.12: Eine einfache Umfrage in Signal

## 3 Fazit und Ausblick

Der Signalbot wurde mit dem Ziel programmiert verschiedene Gruppen um Funktionen zu bereichern, die Signal normalerweise nicht zur Verfügung stellt. Dabei waren Anregungen, Akzeptanz und Nutzverhalten der Anwender\*innen der primäre Fokus. Einzelne, neue Funktionen wurden schnellstmöglich veröffentlicht, um diese von den Nutzer\*innen testen zu lassen. Aus dieser Warte kann zum Projektabschluss behauptet werden, dass das Projekt ein voller Erfolg war. Die meisten Befehle des Bots werden regelmäßig und auf täglicher Basis genutzt.

Der Bot ist bereits ein Teil der Planung und Organisation vom GG eSport Jugendzentrum. Neben den Community-Funktionen werden auch regelmäßige Erinnerungen in die Gruppe geschrieben, um sich in das Wochenprogramm einzutragen oder die Stundenzettel am Ende des Monats abzuschicken.

Zusammen mit der Jugendkirche Düsseldorf ist der `.challenge`-Command entstanden, welcher nun regelmäßig genutzt wird. Es ist zur Tradition geworden, dass die gestellten Herausforderungen ernst genommen und dokumentiert werden, sodass man trotz der Pandemiesituation ein virtuelles Gruppengefühl erzeugen kann.

Zuletzt wurde der Signalbot samt der zugrunde liegenden Technik am 06.08.21 der Firma *IMS Messsysteme GmbH (IMS)* vorgestellt. Es sollte eruiert werden, ob die M2M-Kommunikation aus Abschnitt 2.6 sinnvoll für die Datenübertragung von Sensordaten sein könnte, um eine Basis für Predictive Maintenance zu schaffen. Diese Präsentation baute auf die Bachelorarbeit von Erik Balašćák auf. Die Rückmeldung der IMS war positiv und Signal wird in die Menge der möglichen Technologien zur Datenübertragung im Predictive Maintenance aufgenommen.

### 3.1 Persönliches Fazit

#### 3.1.1 Frederic Aust

Das Projekt war eine interessante Erfahrung, da es vollständig remote absolviert wurde und so gezeigt hat, dass diese Arbeitsweise sehr gut funktionieren kann. Nach den ersten Anlaufschwierigkeiten wurde uns sehr schnell bewusst, dass der Signalbot viel Potential

hat. Insbesondere der Wechsel auf einen daemon, welcher die Antwortzeit von mehreren Minuten zu weniger als eine Sekunde reduziert hat, war der Startschuss für eine rapide Weiterentwicklung des Bots. Diese Entwicklung war mit am faszinierendsten und hat auch immer neue Ideen hervorgebracht. Das super positive Feedback unserer User hat ebenfalls sehr angespornt auch bis in die Abendstunden hinein zu programmieren und den Bot immer weiter auszubauen. Ich denke es gibt noch viele weitere Anwendungszwecke, wie beispielsweise die Abwicklung von Buchungen für Restaurants, Steuerung der Smart-Home-Automatisierung oder als Supportbot für Unternehmen. Als Communitybot wird der Signalbot definitiv weitergepflegt und bekommt noch immer regelmäßig Anfragen für Erweiterungen über den `.request` Befehl. Wann immer Zeit ist arbeite ich an der Umsetzung und wenn es zeitlich passt am liebsten mit geteilten Bildschirm, was die Entwicklungszeit drastisch reduziert hat. Abschließend hat dieses Projekt dazu geführt, dass ich mich in Python erheblich weiter entwickelt habe und auch die vorausschauende Planung von Programmstrukturen und den Modulen fällt mir noch leichter. Für ein neues Projekt wäre ich definitiv aufgeschlossen und hoffe dass wir mit unserem Bot noch mehr Leute begeistern können eigene Instanzen anzulegen.

#### 3.1.2 Philip Maas

Die Arbeit am Signalbot hat unglaublich viel Spaß gemacht. Nachdem das anfängliche Hindernis der Signal-CLI-Installation überwunden war und wir mit dem ersten `.echo`-Kommando eine automatische Reaktion des Bots forcieren konnten ist unser Ideen- und Arbeitspensum regelrecht explodiert. Für mich war dieses Semester ein vielfaches Novum. Dies war mein erstes Semester im Masterstudiengang, mein erstes Coronasemester und mein erstes Semester seit der Ausbildung in dem ich wieder in Vollzeit arbeiten muss. Die generelle Auslastung war fulminant und an manchen Tagen überwältigend. Deshalb wurden viele Abendstunden für die Programmierung des Bots verwendet. Allerdings hat unser Entdeckungsdrang und die regelmäßige positive Rückmeldung der Testpersonen und -Gruppen uns zu einem Endprodukt getrieben auf das Ich sehr stolz bin. Ich freue mich darauf in Zukunft privat den Signalbot weiter zu verbessern und neue Funktionen, ein Refactoring und die Nutzung einer tatsächlichen Datenbank zu implementieren. Ich hoffe, dass der Signalbot auch von kommenden Studierenden genutzt und weiterentwickelt wird.

## 3.2 Ausblick

Obwohl die Arbeit für die beiden Hochschulfächer beendet ist, wird der Signalbot in Zukunft weiterentwickelt. Sobald Nutzer\*innen per `.request` neue Funktionalitäten anfragen werden diese implementiert. Falls der Signalbot in einer höheren Gruppenzahl verwendet wird, müssen für die Geburtstags-, Zitat- und Veranstaltungsfunktionalitäten eine Datenbank verwendet werden. Da an diversen Stellen bereits präferiert mit dem JSON-Format gearbeitet wird, bietet sich eine MongoDB an, aber eine SQL-Variante wäre ebenso denkbar. Beide können mit wenigen Codezeilen über Python angesprochen werden. Da die jetzige Implementierung bereits dynamisch ist, wird der Umzug erleichtert. Zuletzt muss immer wieder Refactoring betrieben werden, um den Code möglichst schlank und lesbar zu halten. In der `betterhelp.json` gibt es bis dato einen Schlüssel `_Func` für jedes Objekt, mit dem in Zukunft die Funktionen für die einzelnen Befehle hinterlegt und aufgerufen werden können.

Da in diesem Projekt mit diversen Anwendungsbereichen gearbeitet wurde, für die Python nicht die primäre Wahl ist, gibt es in den verwendeten Bibliotheken mit Sicherheit Funktionen, die nicht genutzt und somit eigens implementiert wurden. Hier lohnt es sich in regelmäßigen Zeitabständen neue Erkenntnisse, die in Python gemacht wurden, in das Projekt zu übernehmen.



# Abbildungsverzeichnis

1.1	Visualisierung des X3DH [15]	5
1.2	Visualisierung der KDF Kette [17]	7
1.3	Visualisierung des Diffi-Hellman Ratchet Ablauf [18]	8
1.4	Visualisierung des Double Ratchet Ablauf [19]	9
1.5	Visualisierung des Double Ratchet Ablauf Mehrfachversand [20]	10
1.6	CAPTCHA Beispiel	15
1.7	Token in der Firefox Konsole	15
1.8	signal-cli.service	17
1.9	Status signal-cli.service	18
2.1	Statische Texte im Signalbot	20
2.2	Funktionsweise des <code>.yesno</code> -Kommando	21
2.3	Antwort des Bots beim <code>.freegames</code> -Kommando	22
2.4	Auszug aus der <code>gn.json</code> und resultierende Antwort des Bots beim <code>.gn-</code> Kommando	24
2.5	Antwort des Bots beim <code>.nextBirthday</code> -Kommando	27
2.6	Signalbot MQTT Client	28
2.7	Eingesetzte MQTT Architektur	28
2.8	Senden von Testnachrichten via MQTT Explorer	29
2.9	Empfang der MQTT Nachricht via Signal	29
2.10	Empfang des 3D Drucker Status als MQTT Nachricht via Signal	30
2.11	Antwort des Bots beim <code>.tex</code> -Kommando	31
2.12	Eine einfache Umfrage in Signal	32

# Literatur

- [1] Tagesschau. *WhatsApp verschiebt Datenschutz-Änderung*. Datum des Zugriffs: 02.09.2021. URL: <https://www.tagesschau.de/wirtschaft/verbraucher/whatsapp-messenger-facebook-datenschutz-101.html>.
- [2] Till Buecker. *WhatsApp mit neuen Datenschutz-Regeln*. Datum des Zugriffs: 02.09.2021. URL: [tagesschau.de/ausland/amerika/whatsapp-datenschutz-verschiebung-101.html](https://www.tagesschau.de/ausland/amerika/whatsapp-datenschutz-verschiebung-101.html).
- [3] Christine Xuan Mueller. *WhatsApp muss Strafe von 225 Millionen Euro zahlen*. Datum des Zugriffs: 03.09.2021. URL: <https://www.zeit.de/digital/2021-09/whatsapp-irland-bussgeld-rekordstrafe-datenschutz-millions-facebook>.
- [4] Panagiotis Kolokythas. *WhatsApp verwirrt Nutzer mit neuen Nutzungsbedingungen*. Datum des Zugriffs: 03.09.2021. URL: <https://www.pcwelt.de/news/WhatsApp-verwirrt-Nutzer-mit-neuen-Nutzungsbedingungen-10954208.html>.
- [5] Signal Technology Foundation. *Signal Homepage*. Datum des Zugriffs: 02.09.2021. URL: [signal.org](https://signal.org).
- [6] Edward Snowden. *Empfehlung des Signal Messengers*. Datum des Zugriffs: 23.09.2021. URL: <https://twitter.com/Snowden/status/1094963047129628674?s=20>.
- [7] Politico. *Empfehlung des Signal Messengers für Mitglieder der EU-Kommission*. Datum des Zugriffs: 23.09.2021. URL: <https://www.politico.eu/article/eu-commission-to-staff-switch-to-signal-messaging-app/>.
- [8] Signal Technology Foundation. *Signal Foundation Homepage*. Datum des Zugriffs: 23.09.2021. URL: <https://signalfoundation.org>.
- [9] moxie0. *WhatsApp's Signal Protocol integration is now complete*. Datum des Zugriffs: 20.10.2021. URL: <https://signal.org/blog/whatsapp-complete/>.
- [10] Cornelia Möhring - heise online. *WhatsApp-Alternativen: Welche Messenger gibt es?* Datum des Zugriffs: 20.10.2021. URL: <https://www.heise.de/tipps-tricks/WhatsApp-Alternativen-Welche-Messenger-gibt-es-3976153.html/>.
- [11] Ivan Kwiatkowski - kaspersky daily. *Signal – der sichere Messenger für alle, die Wert auf mehr Datenschutz legen*. Datum des Zugriffs: 20.10.2021. URL: <https://www.kaspersky.de/blog/signal-privacy-security/27011/>.

- [12] Katie Benner - The New York Times Nicole Perlroth. *Subpoenas and Gag Orders Show Government Overreach, Tech Companies Argue*. Datum des Zugriffs: 23.09.2021. URL: <https://www.nytimes.com/2016/10/05/technology/subpoenas-and-gag-orders-show-government-overreach-tech-companies-argue.html>.
- [13] Signal Technology Foundation. *X3DH Spezifikation*. Datum des Zugriffs: 20.10.2021. URL: <https://signal.org/docs/specifications/x3dh/>.
- [14] Bruce Schneider. *Applied Cryptography: Protocols, Algorithms and Source Code in C, 20th Anniversary Edition*. ". . .the best introduction to cryptography I've ever seen. . . .The book the National Security Agency wanted never to be published. . . .Wired Magazine. 2015.
- [15] Signal Technology Foundation. *X3DH Aufbau*. Datum des Zugriffs: 20.10.2021. URL: <https://signal.org/docs/specifications/x3dh/X3DH.png>.
- [16] Signal Technology Foundation. *Double Ratchet Spezifikation*. Datum des Zugriffs: 20.10.2021. URL: <https://signal.org/docs/specifications/doubleratchet/>.
- [17] Signal Technology Foundation. *Double Ratchet Aufbau*. Datum des Zugriffs: 20.10.2021. URL: [https://signal.org/docs/specifications/doubleratchet/Set0\\_1.png](https://signal.org/docs/specifications/doubleratchet/Set0_1.png).
- [18] Signal Technology Foundation. *Diffi-Hellman Ratchet Ablauf*. Datum des Zugriffs: 20.10.2021. URL: [https://signal.org/docs/specifications/doubleratchet/Set2\\_1.png](https://signal.org/docs/specifications/doubleratchet/Set2_1.png).
- [19] Signal Technology Foundation. *Double Ratchet Ablauf*. Datum des Zugriffs: 20.10.2021. URL: [https://signal.org/docs/specifications/doubleratchet/Set3\\_2.png](https://signal.org/docs/specifications/doubleratchet/Set3_2.png).
- [20] Signal Technology Foundation. *Double Ratchet Ablauf*. Datum des Zugriffs: 20.10.2021. URL: [https://signal.org/docs/specifications/doubleratchet/Set3\\_3.png](https://signal.org/docs/specifications/doubleratchet/Set3_3.png).
- [21] Signal Technology Foundation. *The XEdDSA and VXEdDSA Signature Schemes*. Datum des Zugriffs: 20.10.2021. URL: <https://signal.org/docs/specifications/xeddsa/>.
- [22] AsamK. *signal-cli GitHub Repository*. Datum des Zugriffs: 26.09.2021. URL: <https://github.com/AsamK/signal-cli/>.
- [23] snapcraft. *rustup - The Rust Language installer*. Datum des Zugriffs: 26.09.2021. URL: <https://snapcraft.io/install/rustup/raspbian>.
- [24] AsamK. *Provide native lib for libsignal*. Datum des Zugriffs: 26.09.2021. URL: <https://github.com/AsamK/signal-cli/wiki/Provide-native-lib-for-libsignal>.

- [25] signalapp. *Library for the Signal Private Group System*. Datum des Zugriffs: 26.09.2021. URL: <https://github.com/signalapp/zkggroup/releases>.
- [26] AsamK. *Building signal-cli*. Datum des Zugriffs: 26.09.2021. URL: <https://github.com/AsamK/signal-cli#building>.
- [27] Signal. *Signal CAPTCHA Generator*. Datum des Zugriffs: 26.09.2021. URL: <https://signalcatchas.org/registration/generate.html>.
- [28] Barbara Eder - c't 14/2021. *Signal aus dem Terminal*. Datum des Zugriffs: 26.09.2021. URL: <https://www.heise.de/select/ct/2021/14/2110907424679785321>.
- [29] AsamK. *DBus service - System bus*. Datum des Zugriffs: 26.09.2021. URL: <https://github.com/AsamK/signal-cli/wiki/DBus-service#system-bus>.
- [30] C653 Labs. *icanhazdadjoke API*. Datum des Zugriffs: 15.08.2021. URL: <https://icanhazdadjoke.com/api>.
- [31] Moritz Kobrna et al. *yesnowtf API*. Datum des Zugriffs: 15.08.2021. URL: <https://yesno.wtf/#api>.
- [32] DeepL GmbH. *DeepL API Docs*. Datum des Zugriffs: 21.10.2021. URL: <https://www.deepl.com/docs-api>.
- [33] Kenneth Reitz. *Python Requests*. Datum des Zugriffs: 15.08.2021. URL: <https://github.com/psf/requests>.
- [34] Leonard Richardson. *Beautiful Soup Dokumentation*. Datum des Zugriffs: 27.08.2021. URL: <https://beautiful-soup-4.readthedocs.io/en/latest/>.
- [35] Sijmen Huizenga Dan Bader. *Schedule Dokumentation*. Datum des Zugriffs: 01.09.2021. URL: <https://schedule.readthedocs.io/en/stable/>.
- [36] MQTT. *MQTT Homepage*. Datum des Zugriffs: 28.09.2021. URL: <https://mqtt.org/>.
- [37] Roger Light. *paho-mqtt Documentation*. URL: <https://pypi.org/project/paho-mqtt/>.
- [38] Thomas Nordquist. *MQTT Explorer Homepage*. Datum des Zugriffs: 29.09.2021. URL: <https://mqtt-explorer.com/>.
- [39] IBM Corporation. *MQTT Client - Subscription wildcards Documentation*. Datum des Zugriffs: 29.09.2021. URL: [https://www.ibm.com/docs/en/ibm-mq/7.5?topic=SSFKSJ\\_7.5.0/com.ibm.mq.javadoc.doc/WMQMqxrcClasses/wildcard.html](https://www.ibm.com/docs/en/ibm-mq/7.5?topic=SSFKSJ_7.5.0/com.ibm.mq.javadoc.doc/WMQMqxrcClasses/wildcard.html).

- [40] Paul Bourke et al. *Four Fours Problem*. Datum des Zugriffs: 03.09.2021. URL: <http://paulbourke.net/fun/4444/>.
- [41] SymPy Development Team. *SymPy Docs: Sympify*. Datum des Zugriffs: 03.09.2021. URL: <https://docs.sympy.org/latest/modules/core.html>.