

Python - DLT - IOTA



Inhalt

1	IoT-Kühlkettenüberwachung mit DLT.....	3
2	DLT - Distributed Ledger Technologie	4
2.1	Grundlagen.....	4
2.2	Welches Hauptproblem löst DLT?	4
3	Was ist IOTA?	5
4	Python und DLT	6
4.1	Installation der IOTA-Client-Bibliothek für Python	6
4.2	Iota-Netze.....	7
4.3	Verbindung mit dem IOTA-Tangle aufnehmen.....	8
4.4	Eine Nachricht auf den Tangle schreiben	10
4.5	Den Tangle live beobachten	12
4.6	Eine Nachricht im Tangle suchen.....	12
4.7	Eine Nachricht mit Inhalt auf den Tangle schreiben	13
4.8	Eine Nachricht mit Testdaten lesen.....	15
4.9	Eine Nachricht im JSON-Format schreiben	16
4.10	Eine Nachricht im JSON-Format lesen	17
5	Wert-Transaktionen	18
5.1	Seed erzeugen	19
5.2	Adressen erzeugen	20
5.3	Konto auffüllen	21
5.4	Kontostand abfragen	22
5.5	UTXO-Wert-Transaktion durchführen	23
6	Verschlüsselung.....	24
6.1	Symmetrische Verschlüsselung mit AES	24
6.2	Passwörter Salzen.....	24
6.3	Padding.....	25
6.4	Die Bibliothek „pycryptodome“	25
6.5	Einen Text verschlüsseln.....	26
6.6	Einen verschlüsselten Text wieder entschlüsseln	27
6.7	Eine verschlüsselte Nachricht auf den Iota-Tangle schreiben.....	28
6.8	Eine verschlüsselte Nachricht vom Iota-Tangle lesen	29
7	Projektstufe 1	30
7.1	Scenario	30
7.2	Technische Spezifikationen	30

7.3	Aufgabenstellung.....	31
8	Projektstufe 2.....	32
8.1	Temperaturüberwachung der Kühlstationen	32
8.2	Verschlüsselung der Daten auf dem Tangle	33
8.3	Reparationszahlung bei Verletzung der Kühlkette	34
8.4	Abgabe	34
9	ECLASS.....	35
10	MQTT.....	36
10.1	Eine Nachricht mit Temperaturdaten per MQTT lesen.....	36
11	DLT-Anwendungsszenarien	37
12	DLT-Detailwissen	38
12.1	Permissionless DLT	38
12.2	Merkmale	39
12.2.1	Dezentralität.....	39
12.2.2	Unveränderlichkeit	39
12.2.3	Programmiertes Vertrauen.....	39
12.2.4	Doppelausgabenproblem ist gelöst	39
12.2.5	Transparenz	39
12.2.6	Sofortige Transaktionen	39

1 IoT-Kühlkettenüberwachung mit DLT



Ein Bio-Döner möchte die Kühlkette seines Fleisches überwachen.

Programmieren Sie eine

IoT-DLT-Lösung

in Python

mit IOTA!



Schauen Sie folgende Film:

<https://youtu.be/L250oRM8po>



Für die Lösung dieser Aufgabe müssen wir folgende Fragen/Teilaufgaben klären:

1. Was ist Distributed Ledger Technologie?
2. Was ist IOTA?
3. Wie schreibe ich Daten auf den IOTA-Tangle?
4. Wie lese ich Daten vom IOTA-Tangle?

2 DLT - Distributed Ledger Technologie

Schauen Sie als Einleitung in das Thema folgenden Vortrag:

<https://www.youtube.com/watch?v=pUihIdE9os>



2.1 Grundlagen

Der Begriff Distributed-Ledger-Technologie (englisch für Technik verteilter Kassenbücher od. Hauptbücher) ist eine Datenbankarchitektur, die es den Besitzern digitaler Güter ermöglicht, diese von Peer zu Peer zu übertragen und zu dokumentieren. Jede Übertragung in einem DLT wird als Datensatz in einem Distributed-Ledger (Datenbank) gespeichert, diese Datenbank ist in allen Nodes eines Netzwerks gespeichert.

Im Gegensatz zum klassischen Ansatz, bei dem ein Kassenbuch in der Regel von nur einer Instanz verwaltet wird, werden hier dezentral beliebig viele gleichgestellte Kopien des Kassenbuchs von unterschiedlichen Parteien gespeichert. Durch geeignete Maßnahmen wird dafür gesorgt, dass neu hinzuzufügende Transaktionen in allen Kopien des Kassenbuchs übernommen werden und dass es zu einer Übereinkunft (Konsensus) über den jeweils aktuellen Stand des Kassenbuchs kommt.

Es wird auch von dezentral geführten Kontobüchern oder Transaktionsdatenbanken gesprochen. Die Technik gilt als wegweisend für die Verwaltung von Daten im Internet ohne auf eine Eigentümerplattform zurück greifen zu müssen.

Die Distributed-Ledger-Techniken unterscheiden sich durch die Art, wie die vernetzten Computer zu einer Vereinbarung kommen (Konsensus Protokolle), etwa durch „Proof of Work“, durch „Proof of Stake“ und weiteren Verfahren oder Kombinationen.

2.2 Welches Hauptproblem löst DLT?

Problem: Wenn Daten in Eigentümer geführten Datenbanken gespeichert werden, ist es schwierig, diese Daten mit anderen zu teilen, ohne dass sie geändert werden können oder in anderen Datenbanken verloren gehen.

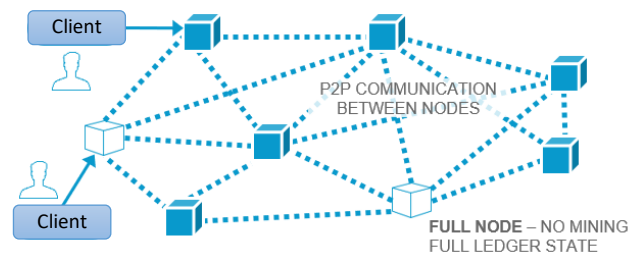
Lösung: DLT schafft eine einzige Wahrheit, auf die alle Teilnehmer vertrauen können. Wenn Daten zu einem verteilten Kassenbuch hinzugefügt werden, kann jeder mit einer Internetverbindung darauf zugreifen, indem er sich mit einem Node im Netzwerk verbindet.

Quelle: <https://iota-einsteiger-guide.de/grundlagen/archiv-distributed-ledger-technologie-dlt/>

3 Was ist IOTA?

Das Ziel der IOTA Foundation ist es, eine Vertrauensschicht (engl. Trust Layer) für das Internet of Everything (IoE) zu erschaffen, die es Geräten im IoE ermöglicht unveränderlich Daten und Werte untereinander gebührenfrei auszutauschen. IOTA strebt in Zusammenarbeit mit der Industrie und der Object-Management-Group, eine Standardisierung ihres Kommunikationsprotokolls an. Mit einer hohen Interoperabilität wird IOTA das "Ledger Of Everything" sein, dessen Infrastruktur auch von externen Anwendungen ohne Erlaubnis genutzt werden kann.

IOTA ermöglicht eine schnelle, manipulationssichere und dezentrale Übertragung von Werten und Daten über viele Nodes, dabei werden Werte- und Daten-Transaktionen grundsätzlich unterschiedlich gehandhabt, während Werte-Transaktionen von Full-Nodes validiert werden müssen, werden Daten-Transaktionen direkt bestätigt und sind notariert.



Jetzt wird sich der ein oder andere Fragen, warum benötige ich für eine reine Daten-Transaktion die IOTA Distributed Ledger Technologie, ich kann die Daten doch einfach verschlüsseln, signieren und via TCP/IP versenden.

Nun, abgesehen davon, dass „Man in the Middle“ Angriffe möglich wären, beweisen signierte Daten nur, dass die Daten von Ihnen kommen. Es erlaubt mir weder zu beweisen, wann Sie sie gesendet haben, noch ob Sie die gleichen Daten an alle gesendet haben. Sie könnten eine bestimmte Information an eine Person und eine andere Information an eine andere Person senden. Signaturen allein werden niemanden vor solchen Dingen schützen.

Mithilfe der „Notarisierung“ kann bewiesen werden, dass ein elektronisches Dokument in einer bestimmten Form zu einem bestimmten Zeitpunkt existiert hat und seit der Erstellung nicht verändert wurde. Bei der Erstellung einer Notarisierung wird ein eindeutiger Hash (Fingerabdruck) eines Dokumentes berechnet und gemeinsam mit einem Zeitstempel im IOTA-Ledger (Tangle) unveränderbar gespeichert. Falls zu einem späteren Zeitpunkt verifiziert werden soll, dass das betreffende Dokument zum behaupteten Zeitpunkt existiert hat und/oder nicht verändert wurde, werden die Daten aus dem Tangle abgerufen und mit den vorliegenden Informationen verglichen.

Unterm Strich bedeutet dies, dass es nicht nur darum geht, dass niemand die Daten während der Übertragung manipuliert, sondern auch darum, dass der Empfänger diese Daten nicht manipuliert. Beispiel: Ein Sensor (mit IDoT Chip) hat einige Werte gemessen/erfasst und versendet diese Daten über den IOTA-Tangle, welcher den Hash dieser Daten speichert. Wenn diese Daten später verkauft werden sollen, kann dieser Hash als Nachweis vorgelegt werden und dem Käufer anhand des Tangle beweisen, dass die Daten vom Sensor im Nachhinein nicht verändert wurden. Die IOTA Technologie (Tangle) fungiert also wie eine Art Fingerabdruck, mit ihm können alle gesendeten Daten verifiziert werden.

Quelle: <https://iota-einsteiger-guide.de/einfuehrung/archiv-was-ist-iota-eine-zusammenfassung/>

4 Python und DLT

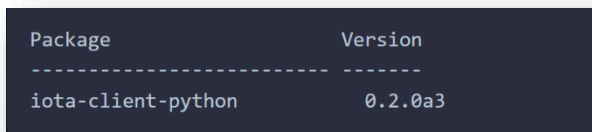
4.1 Installation der IOTA-Client-Bibliothek für Python

Für die Interaktion mit dem IOTA-Tangle über Python muss eine Iota-Client-Bibliothek installiert werden. Hierfür sind vier Schritte notwendig:

1. Eingabeaufforderung (cmd oder powershell) **als Administrator** starten
2. Tool zur Verarbeitung von „wheel“-Dateien installieren mit: `py -m pip install wheel`
3. Wheel-Datei mit Python-Bibliothek herunterladen:
<https://nightly.link/iotaledger/iota.rs/workflows/python-bindings-publish/production>
Die Datei muss entpackt werden.
Wechseln Sie in das Verzeichnis, in welches Sie die Datei entpackt haben.
4. Wheel-Datei mit Python-Bibliothek installieren: `py -m pip install <wheel_file>`
Hinweis: Unbedingt den kompletten Dateinamen inkl. „.whl“ angeben.

Eine ausführliche Installationsanleitung finden Sie hier: https://wiki.iota.org/iota.rs/getting_started/python

Die erfolgreiche Installation kann mit dem Befehl `py -m pip list` überprüft werden.



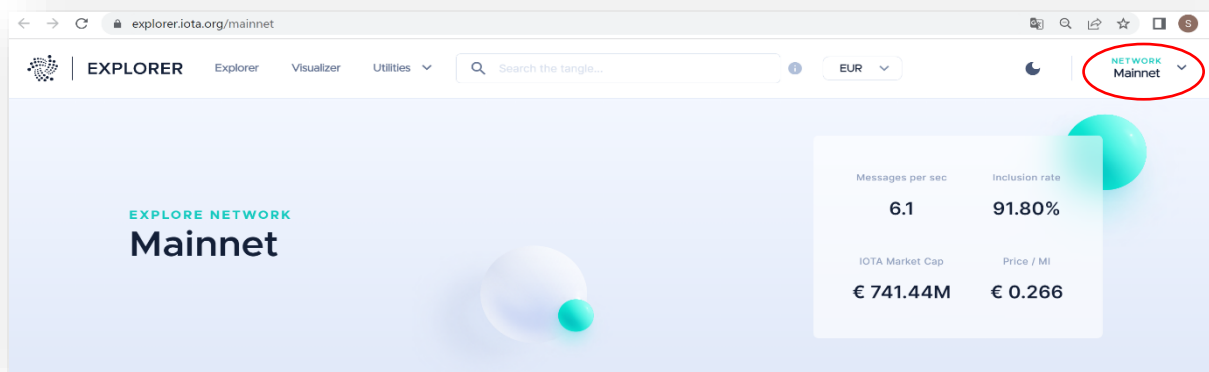
Package	Version
-----	-----
iota-client-python	0.2.0a3

4.2 Iota-Netze

Die Iota-Foundation betreibt aktuell folgende Netzwerke:

- | | |
|--------------------------------|---|
| • Mainnet (Chrysalis) | Aktuelles Hauptnetz mit handelbaren Iota-Tokens (Version Chrysalis) |
| • Chrysalis Devnet (Chrysalis) | Testnetz mit wertlosen Iota-Tokens (Version Chrysalis) |
| • Shimmer (Stardust) | Staging Netz mit handelbaren Shimmer-Tokens (Version Stardust) |
| • Testnet (Stardust) | Testnetz mit wertlosen Shimmer-Tokens (Version Stardust) |
| • Legacy Mainnet (Legacy) | Altes Mainnet vor dem Chrysalis-Update |

Sie können sich die verschiedenen Netze unter <https://explorer.iota.org> anschauen.



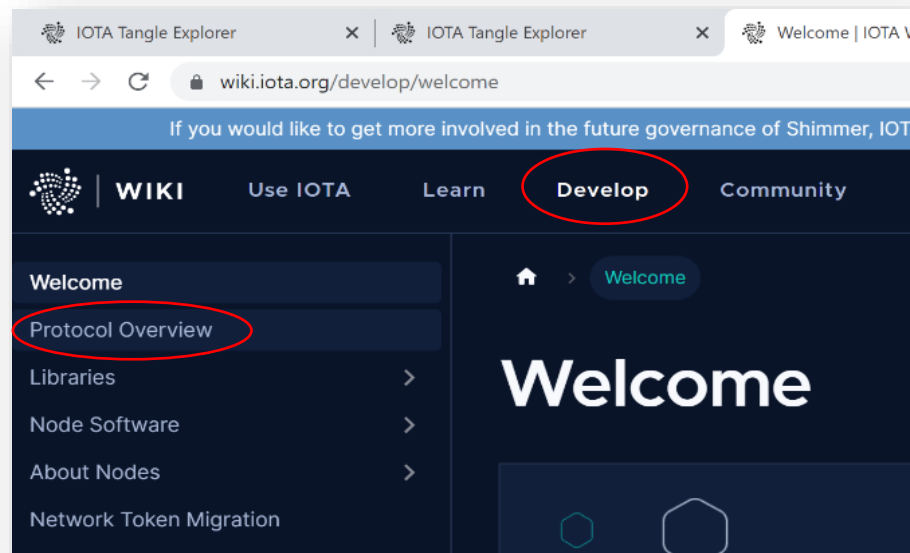
Aufgabe

Wie viele Nachrichten werden im Mainnet aktuell pro Sekunde bestätigt? _____

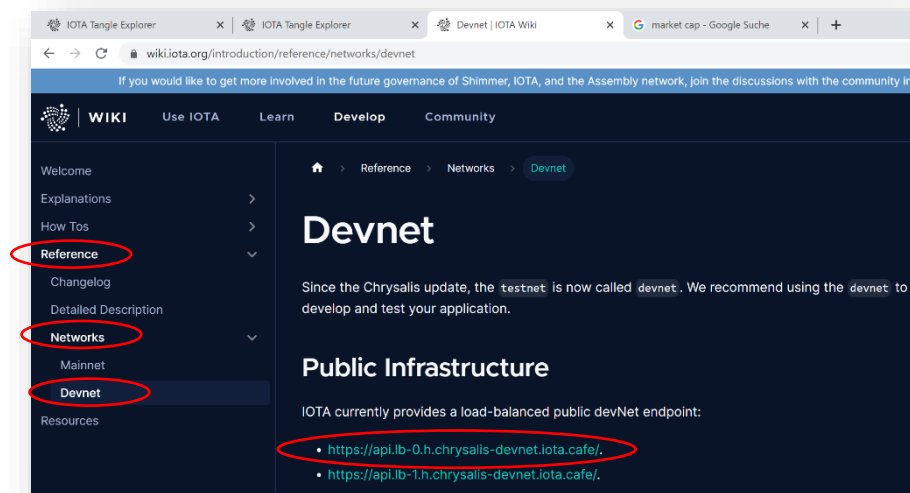
4.3 Verbindung mit dem IOTA-Tangle aufnehmen

Für unsere Testprogramme nutzen wir das „Chrysalis-Devnet“.

Für die Kontaktaufnahme benötigen wir die URL zu einem bestimmten Knoten des Devnets oder die URL zu einem Loadbalancer für das Devnet, der uns dann dynamisch zu einem entsprechenden Knoten weiterleitet.



Die benötigten URLs sind im Iota-Wiki unter „Develop“ => „Protocol Overview“ => dokumentiert:



Verbinden Sie sich zunächst mit einem Knoten und zeigen Sie Informationen zu dem Knoten an.

```
import iota_client

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe'

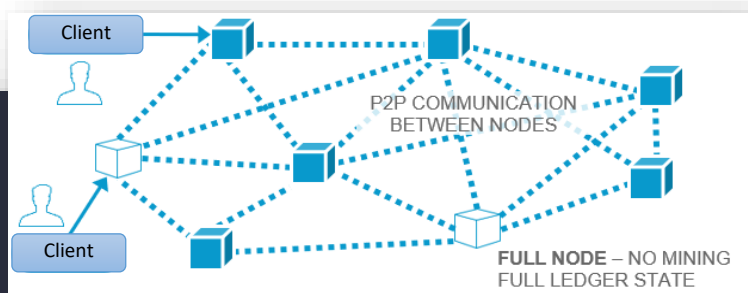
# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Knoteninfos ausgeben
print(client.get_info())
```

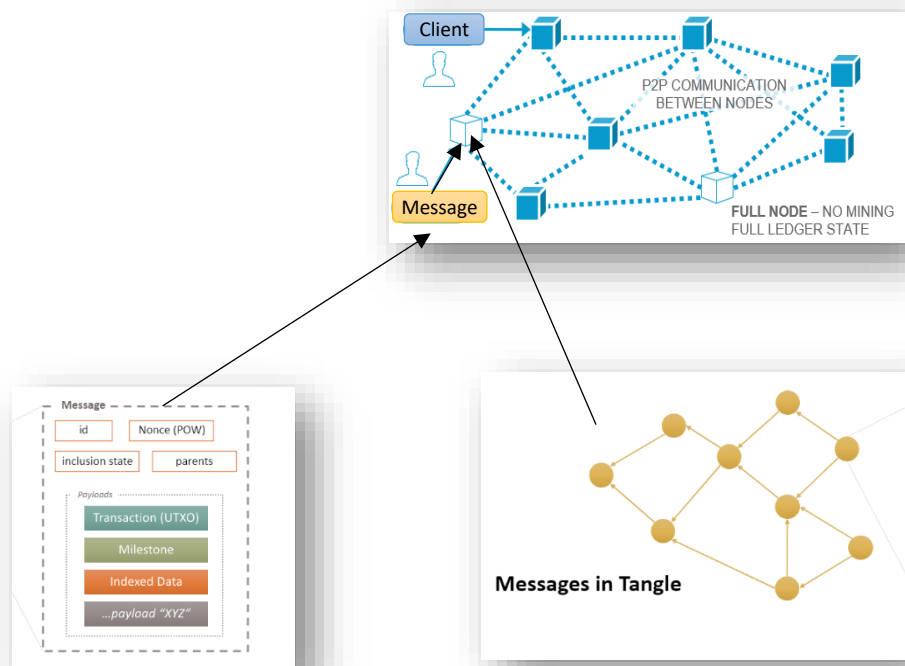
Eine Anleitung finden Sie hier: https://wiki.iota.org/iota.rs/examples/get_info

Beispielinfo zu einem Knoten:

```
{
  "nodeinfo": {
    "name": "HORNET",
    "version": "0.6.0-alpha",
    "is_healthy": true,
    "network_id": "migration",
    "bech32_hrp": "atoi",
    "min_pow_score": 100,
    "messages_per_second": 4.2,
    "referenced_messages_per_second": 4.1,
    "referenced_rate": 97.61904761904762,
    "latest_milestone_timestamp": 1618139001,
    "latest_milestone_index": 7092,
    "confirmed_milestone_index": 7092,
    "pruning_index": 0,
    "features": [
      "Pow"
    ]
  },
  "url": "https://api.lb-0.h.chrysalis-devnet.iota.cafe"
}
```



4.4 Eine Nachricht auf den Tangle schreiben



Mit den folgenden Befehlen können Sie eine Message auf den IOTA-Tangle schreiben:

```
import iota_client

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.thin-horner-0.h.chrysalis-devnet.iota.cafe'

# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Nachricht auf den Tangle schreiben
message = client.message()

# Nachricht ausgeben
print(message)
```

Eine Anleitung finden Sie hier: https://wiki.iota.org/iota.rs/examples/simple_message

Als Ergebnis wird die Nachricht im folgenden Format erzeugt und ausgegeben.

```
{
  "message_id": "e2daa4c6b012b615becd6c12189b2c9e701ba0d53b31a15425b21af5105fc086",
  "network_id": 7712883261355838377,
  "parents": [
    "0e2705ce50fec88f896663d4b7d562e74cbcfdd951ac482b1f03cfa5f27396d7",
    "0f5a0b2041766127c3f3bff2dd653b450b72e364765fcc805a40423c59ed01f9",
    "20635b30aee437575d7e6abdf6629eec80543bee30848b0abdda2200fc11a977",
    "da97cd6cfcbb854b8fd3f064c8459c5c9eae80dbd5ef594a3e1a26dcb8fc078c"
  ],
  "payload": "None",
  "nonce": 2305843009213869242
}
```

Die Zeile "payload": "None" sagt aus, dass die Nachricht keinen Inhalt/Wert enthält.

Speichern Sie ihre ausgegebene „Message-ID“ ab, um sie später über Tangle-Explorer im Tangle zu suchen.

4.5 Den Tangle live beobachten

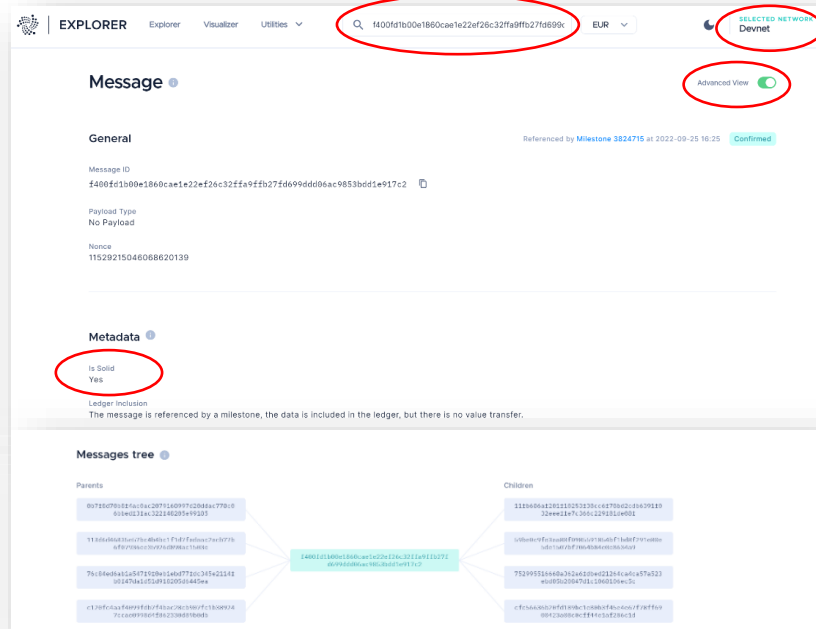
Über folgenden Link: <https://explorer.iota.org/devnet/visualizer/> können Sie die Entwicklung des IOTA-Tangles (bzw. eines Ausschnitts davon) live beobachten.

Der Tangle ist die Datenbank, in der die Nachrichten abgelegt und miteinander verknüpft sind. Der Tangle ist sozusagen das Kassenbuch im DLT. Der Tangle liegt gleichermaßen auf allen Knoten.



4.6 Eine Nachricht im Tangle suchen

Über das Suchfeld und Ihre Message-ID können Sie Ihre Nachricht im Tangle anzeigen.



Da die Nachricht den Status „Solid“ besitzt, ist sie durch die DLT bestätigt und kann nicht mehr verändert werden.

4.7 Eine Nachricht mit Inhalt auf den Tangle schreiben

Bitte überlegen Sie sich ein persönliches dreistelliges Kürzel, welches Sie statt „RSE“ benutzen.

Mit den folgenden Befehlen können wir „Testdaten“ unter dem Suchindex „RSE-Test“ auf den Tangle schreiben.

```
import iota_client

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe'

# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Message auf den Tangle schreiben
message = client.message(index = "RSE-Test", data_str = "Testdaten")

# Message ausgeben
print(message)
```

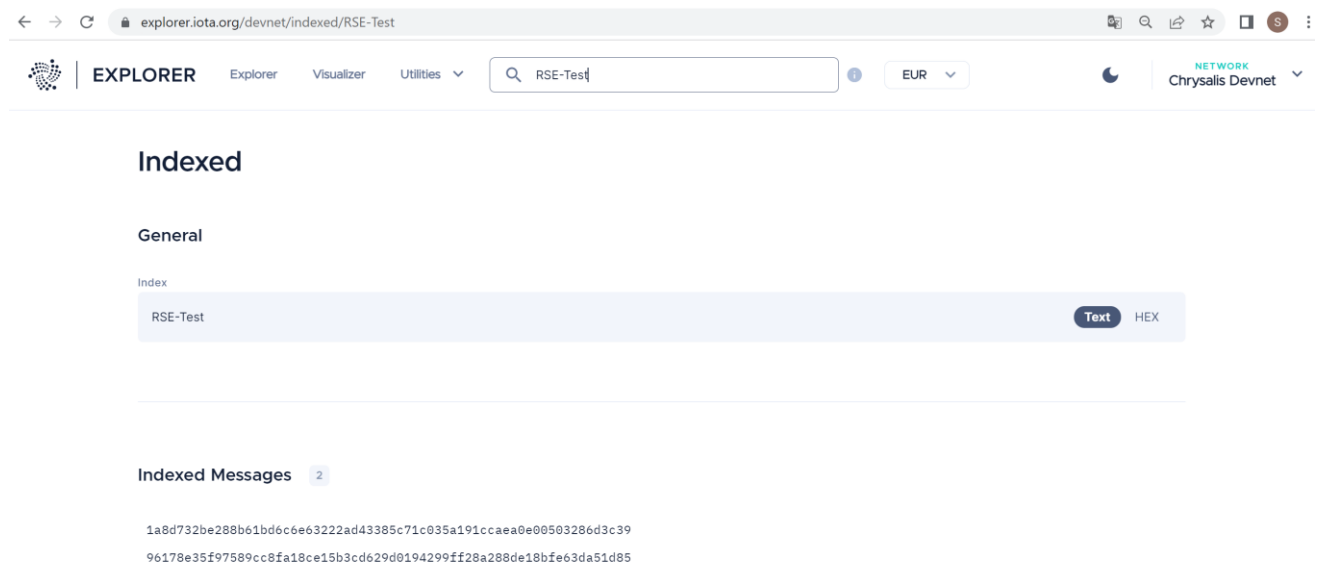
Eine Anleitung finden Sie hier: https://wiki.iota.org/iota.rs/examples/data_message

Die gesendete Nachricht sieht folgendermaßen aus:

```
{
  'message_id': 'c71ac44e7fc0eeaa2008aeaf49628a704a706ad9fb9141712e1e513956d8e487',
  'network_id': 6514788332515804015,
  'parents': [
    '8af079c56003713913b00503438fe131b5791b4f2bc858137942002f5c70d892',
    'c92ae1c87b0e980f7f0009313cb176a3e2ed97c7c815c551d02d9667432ee864',
    'caddc645cc8065a725e97f402539e6a551be92ee14f28fa6dfc1ef5ae05b3baf',
    'cdd0cba2d6a3de47a40e412772d4a113fdcdc1fb13ebf113926be284f094a9ef'
  ],
  'payload': {
    'transaction': None,
    'milestone': None,
    'indexation': [{
      'index': '5253452d54657374',
      'data': [84, 101, 115, 116, 100, 97, 116, 101, 110]
    }],
    'receipt': None,
    'treasury_transaction': None
  },
  'nonce': 11529215046068484309
}
```

Weitergehende Informationen gibt es hier: <https://wiki.iota.org/learn/about-iota/data-transfer>

Auch diese Nachricht können wir mit dem Visualizer : <https://explorer.iota.org/devnet/visualizer/> auf dem Tangle wiederfinden. Die Nachricht kann entweder anhand der „Message-ID“ oder des Index „RSE-Test“ gesucht werden. Bei der Indexsuche werden alle Nachrichten zu diesem Index angezeigt.



The screenshot shows the IOTA Explorer interface. The browser address bar displays `explorer.iota.org/devnet/indexed/RSE-Test`. The page title is "EXPLORED". The navigation menu includes "Explorer", "Visualizer", and "Utilities". A search bar contains the text "RSE-Test". The currency is set to "EUR". The network is "Chrysalis Devnet".

Indexed

General

Index

RSE-Test Text HEX

Indexed Messages 2

1a8d732be288b61bd6c6e63222ad43385c71c035a191ccaea0e00503286d3c39

96178e35f97589cc8fa18ce15b3cd629d0194299ff28a288de18bfe63da51d85

4.8 Eine Nachricht mit Testdaten lesen

```
import iota_client

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe'

# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Messages zu einem Index suchen
mlist = client.find_messages(indexation_keys=['RSE-Test'])

# Schleife über alle gefundenen Messages
for message in mlist:
    # Daten des ersten Eintrages => Liste mit Dezimalwert der Einzelzeichen
    pl = message['payload']['indexation'][0]['data']

    # Dezimalwerte in String mit ASCII-Zeichen umwandeln
    s = ''.join(chr(val) for val in pl)

    print('RSE-Test: ' + s)
    print('-----')
```


4.9 Eine Nachricht im JSON-Format schreiben

```
import iota_client
from datetime import datetime
import json

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe'

# Sensordaten als dict anlegen
payload = {
    'sensor-id': 'TS1',
    'timestamp': datetime.now().strftime("%d.%m.%Y %H:%M:%S"),
    'temperature': 7.2
}

# dict als JSON-String ablegen
jspayload = json.dumps(payload)

# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Message auf den Tangle schreiben
message = client.message(index="RSE-TS1", data_str = jspayload)

# Message ausgeben
print(message)
```

4.10 Eine Nachricht im JSON-Format lesen

```
import iota_client
import json

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe'

# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Messages zu einem Index suchen
mlist = client.find_messages(indexation_keys=["RSE-TS1"])

for message in mlist:
    pl = message['payload']['indexation'][0]['data'] # Schleife über alle gefundenen Messages
    s = ''.join(chr(val) for val in pl) # Daten => Liste mit Dezimalwert der Einzelzeichen
    d = json.loads(s) # Dezimalwerte in String umwandeln
    for key in d: # Dictionary aus JSON-String erzeugen
        print(key + ': ' + str(d[key])) # Alle Einträge des Dictionary durchgehen
    print('-----') # Schlüssel und Schlüsselwert ausgeben
```

5 Wert-Transaktionen

In den bisherigen Übungen haben wir gelernt, wie man Nachrichten auf den Iota-Tangle schreibt und vom Iota-Tangle liest. Ein besonderer Nachrichtentyp ist die Wert-Transaktion. Hier wird ein besonderes Verfahren (unspent transaction output – UTXO) zur Übertragung von „Iota“ angewendet.

Für eine Wert-Transaktion ist die gesendete Nachricht vom Typ UTXO und enthält einen Betrag, sowie eine Absender- und eine Zieladresse. Der angegebene Betrag wird der Iota-Zieladresse gutgeschrieben und der Iota-Absenderadresse abgezogen. Eine Wert-Transaktion funktioniert vergleichbar mit einer Banküberweisung zwischen zwei Konten.

Ziel der nächsten Übung ist es, eine Million Iotas (1 MIOTA) von Anna an Bob zu „überweisen“. Damit wir diese Wert-Transaktion durchführen können, müssen wir folgende Schritte ausführen:

- **Seed** für die „Konten“ von Anna und Bob erzeugen
- **Iota-Adressen** für Anna und Bob aus den Seeds erzeugen
- Annas Konto über eine **Faucet-Webseite** aufladen und **Kontostände prüfen**
- **UTXO-Wert-Transaktion** von Anna zu Bob durchführen und **Kontostände prüfen**

Eine ausführliche Anleitung finden Sie hier: https://wiki.iota.org/iota.rs/examples/generate_seed/

5.1 Seed erzeugen

Als erstes benötigen Anna und Bob ein Konto. Dieses wird im Iota-Tangle durch einen „Seed“ repräsentiert.

Ein IOTA Seed ist ein privater Schlüssel, der verwendet wird, um Zugang zu einem IOTA-Konto (einer IOTA-Wallet) zu erhalten. Er dient als "Hauptschlüssel" für IOTA-Transaktionen und sollte sorgfältig aufbewahrt werden, da er nicht wiederhergestellt werden kann, wenn er verloren geht. Der Seed ist in der IOTA-Technologie einzigartig und ermöglicht es Benutzern, ihre IOTA-Token sicher zu verwalten und Transaktionen durchzuführen. Es ist wichtig zu beachten, dass ein Seed nicht mit einem Passwort oder einer PIN verwechselt werden sollte, da es ein komplexer Schlüssel ist, der allein den Zugang zu einer IOTA-Wallet gewährt.

Wir müssen jeweils einen Seed für Anna und Bob erzeugen und sichern.

```
# Bibliotheken
import os
import hashlib

# Seed mit 64 Hex-Stellen erzeugen => 264 Bit
# 2^64 = 18.446.744.073.709.551.616 = 18.446 Billiarden verschiedene Werte
rnd_seed = hashlib.sha256(os.urandom(256)).hexdigest()

# Ausgabe
print ('-----')
print ("Seed: ", rnd_seed)
print ('-----')

# Seed für Anna und Bob merken
A_IOTA_SEED_SECRET = "6b403b2dcf927ca42d15bb90480e2cad69e0e22400890ce399bab4ebf91537d8"
B_IOTA_SEED_SECRET = "5971faee8c4d762eed1ea2d4496635cbc22d4a22eecfca5dc6b2ad25316d9a82"
```

Starten Sie das Programm zweimal und merken Sie sich jeweils den Seed, indem Sie ihn in den Programmcode kopieren!

Ausgabe:

```
-----
Seed:  6b403b2dcf927ca42d15bb90480e2cad69e0e22400890ce399bab4ebf91537d8
-----
```

```
-----
Seed:  5971faee8c4d762eed1ea2d4496635cbc22d4a22eecfca5dc6b2ad25316d9a82
-----
```

5.2 Adressen erzeugen

Aus dem Seed können nun beliebig viele Iota-Adressen erzeugt werden. Wert-Transaktionen sind immer an eine Ziel- und Quell-Adresse gebunden.

```
# Bibliotheken
import os
import iota_client

# Seed setzen
A_IOTA_SEED_SECRET = "6b403b2dcf927ca42d15bb90480e2cad69e0e22400890ce399bab4ebf91537d8"
B_IOTA_SEED_SECRET = "5971faee8c4d762eed1ea2d4496635cbc22d4a22eefca5dc6b2ad25316d9a82"
IOTA_SEED_SECRET = A_IOTA_SEED_SECRET

# Client-Objekt erzeugen
client = iota_client.Client()

# Iota-Adresse erzeugen
address_changed_list = client.get_addresses(
    seed=IOTA_SEED_SECRET,
    account_index=0,
    input_range_begin=0,
    input_range_end=1,
    get_all=True
)

#Ausgabe
print ('-----')
print("Iota-Adresse: ", address_changed_list)
print ('-----')

# Adressen merken (False=Public-Address, True=Internal Address)
A_Adresse = [('atoi1qz7qm2xt59fa6yh949xucynm4vu24clgh7x7j93une5k39x2h20pw7j4vvvg', False),
              ('atoi1qqlv4wjddv18dd8fr8x5l5r0u4te3pm7svc0yg5lhwdejhxqsrlds45gsav', True)]

B_Adresse = [('atoi1qzlhqj9fu3ce7np6rn052kfushw2d5c5ff62he271ldeas319ux46pnzjcm', False),
              ('atoi1qr8sn9v1mygqawep4lulfq84gehjqk8nxrlzg5ghgd356c3992e6gxxg0qt7', True)]
```

Starten Sie das Programm einmal mit dem Seed von Anna und einmal mit dem Seed von Bob.
Merken Sie sich die beiden Ausgaben im Programmcode.

Ausgabe:

```
-----
Iota-Adresse: [('atoi1qz7qm2xt59fa6yh949xucynm4vu24clgh7x7j93une5k39x2h20pw7j4vvvg', False), ('atoi1qqlv4wjddv18dd8fr8x5l5r0u4te3pm7svc0yg5lhwdejhxqsrlds45gsav', True)]
-----
```

```
-----
Iota-Adresse: [('atoi1qzlhqj9fu3ce7np6rn052kfushw2d5c5ff62he271ldeas319ux46pnzjcm', False), ('atoi1qr8sn9v1mygqawep4lulfq84gehjqk8nxrlzg5ghgd356c3992e6gxxg0qt7', True)]
-----
```

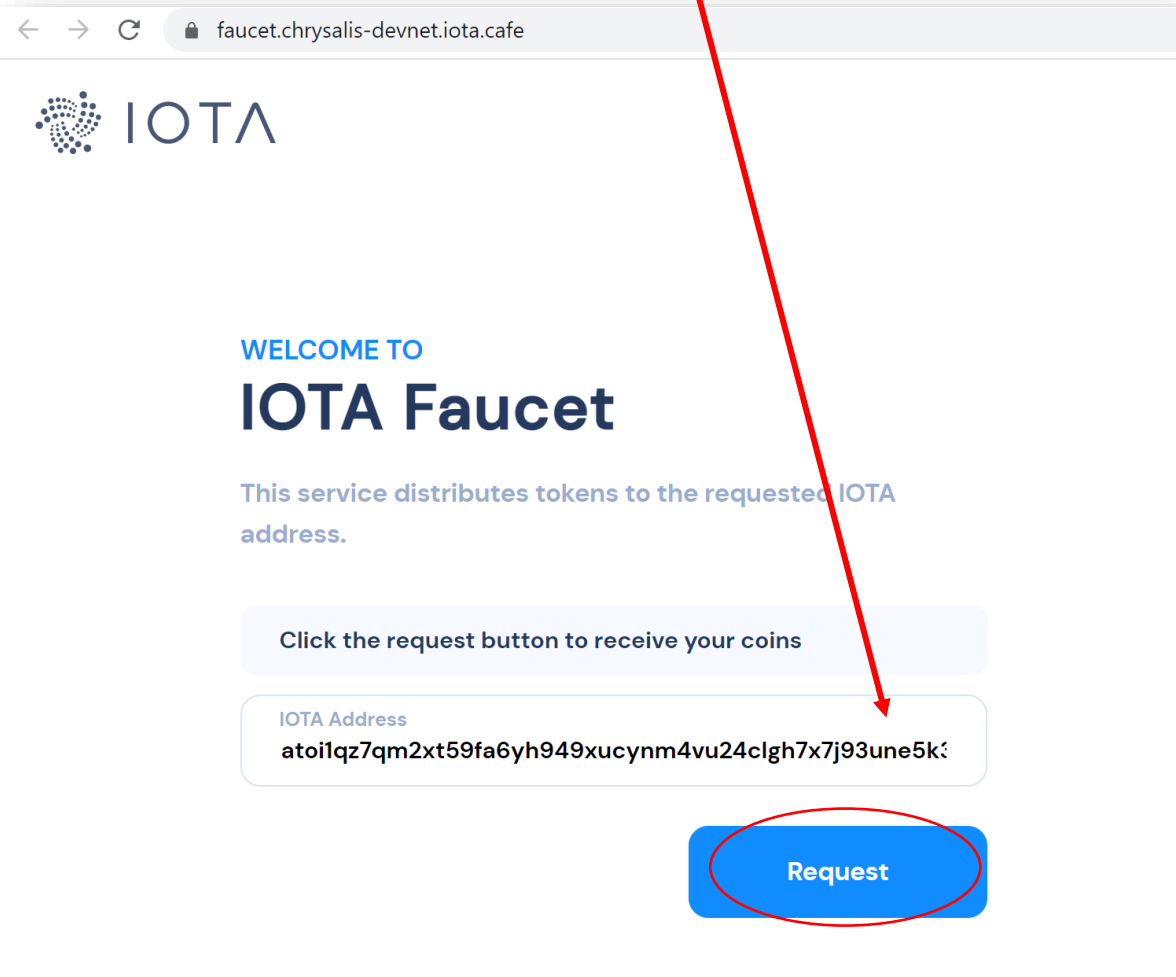
5.3 Konto auffüllen

Da unsere Wert-Transaktionen im DevNet stattfinden, können wir das Konto von Anna umsonst mit Iota-Token aufladen.


Kopieren Sie die erzeugte öffentliche Adresse von Anna (obere Adresse, hinter der „False“ steht) und laden Sie über die folgende Webseite Iota-Token auf das Konto:

<https://faucet.chrysalis-devnet.iota.cafe/>

```
20
27 # Adressen merken (False=Public-Address, True=Internal Address)
28 A_Adresse = [('atoi1qz7qm2xt59fa6yh949xucynm4vu24clgh7x7j93une5k39x2h20pw7j4vvg', False),
29              | ('atoi1qqlv4wjdv18dd8fr8x515r0u4te3pn7svc0yg51hwdejhxqsrlds45gsav', True)]
30
31 B_Adresse = [('atoi1qz1hqj9fu3ce7np6rn052kfushw2d5c5ff62he271ldeas319ux46pnzjcm', False),
32              | ('atoi1qr8sn9vlmygqawep41ulfq84gehjqk8nrlzg5ghgd356c3992e6gxg0qt7', True)]
33
34
```



← → ↻ 🔒 faucet.chrysalis-devnet.iota.cafe

 IOTA

WELCOME TO
IOTA Faucet

This service distributes tokens to the requested IOTA address.

Click the request button to receive your coins

IOTA Address
atoi1qz7qm2xt59fa6yh949xucynm4vu24clgh7x7j93une5k39x2h20pw7j4vvg

Request

5.4 Kontostand abfragen

Überprüfen Sie als nächstes den Kontostand für die Seeds von Anna und Bob ab.

```
# Bibliotheken
import os
import iota_client

# Seeds
A_IOTA_SEED_SECRET = "6b403b2dcf927ca42d15bb90480e2cad69e0e22400890ce399bab4ebf91537d8"
B_IOTA_SEED_SECRET = "5971faee8c4d762eed1ea2d4496635cbc22d4a22eecfca5dc6b2ad25316d9a82"

# Client-Objekt erzeugen
client = iota_client.Client()

# Kontostand für die Seeds abfragen
A_balance = client.get_balance(seed=A_IOTA_SEED_SECRET, account_index=0,
initial_address_index=0)
B_balance = client.get_balance(seed=B_IOTA_SEED_SECRET, account_index=0,
initial_address_index=0)

# Ausgabe
print ('-----')
print("Kontostand Anna: ", A_balance/1000000, " MIOTA")
print ('-----')
print("Kontostand Bob:  ", B_balance/1000000, " MIOTA")
print ('-----')
```

Ausgabe:

```
-----
Kontostand Anna:  10.0  MIOTA
-----
Kontostand Bob:   0.0  MIOTA
-----
```

5.5 UTXO-Wert-Transaktion durchführen

Im nächsten überweisen wir mit einer UTXO-Wert-Transaktion 1 MIOTA von Anna zu Bob.

```
# Bibliotheken
import os
import iota_client

# Seed von Anna
A_IOTA_SEED_SECRET = "6b403b2dcf927ca42d15bb90480e2cad69e0e22400890ce399bab4ebf91537d8"

# IOTA-Adresse von Bob
B_Adresse = [('atoi1qzlhqj9fu3ce7np6rn052kfushw2d5c5ff62he271ldeas3l9ux46pnzjcm', False),
              ('atoi1qr8sn9v1mygqawep4lu1fq84gehjqk8nxrlzg5ghgd356c3992e6gxxg0qt7', True)]

# Client-Objekt erzeugen
client = iota_client.Client()

# Überweisung durchführen
message = client.message(
    seed=A_IOTA_SEED_SECRET,
    outputs=[
        {
            'address': B_Adresse[0][0],
            'amount': 1_000_000
        }
    ]
)

# Ausgabe
print ('-----')
print(message)
print ('-----')
```

Ausgabe:

```
{'message_id': 'e2ce3b31b8115fe5d9d3c0b39a06f5e5358e134cc0648271d86570e8d2746ca', 'network_id': 651478833251580
4015, 'parents': ['1c8e1c0128d0524a86d033a0e63d1426e0c30937f6674604add194390ebcb1b9', '8030f5ffc7267a351a316dff9
60e1b04e0e98186e1428b00d864316ed3406c7b', 'a13833ad535ba7054e335bcb65537073c86dbd46e2f779ea0fcbf64894389109', 'b
03df867ea2b3ad77304f6340df742e4cf1dd10102ce119fe4c1332916cdf14a'], 'payload': {'transaction': [{'essence': {'inp
uts': [{'transaction_id': 'b79f4d156c907da868837ded952a52d9757891292444a907069c7812526b4690', 'index': 1}], 'out
puts': [{'signature_locked_single': {'address': 'atoi1qz7qm2xt59fa6yh949xucynm4vu24clgh7x7j93une5k39x2h20pw7j4vv
g', 'amount': 9000000}, 'signature_locked_dust_allowance': None}, {'signature_locked_single': {'address': 'atoi
qzlhqj9fu3ce7np6rn052kfushw2d5c5ff62he271ldeas3l9ux46pnzjcm', 'amount': 1000000}, 'signature_locked_dust_allowan
ce': None}], 'payload': None}, 'unlock_blocks': [{'signature': {'public_key': [62, 133, 221, 184, 73, 97, 35, 14
, 46, 199, 178, 101, 10, 204, 101, 219, 66, 160, 88, 88, 109, 205, 244, 66, 243, 223, 231, 88, 0, 212, 157, 109]
, 'signature': [238, 116, 66, 24, 38, 224, 16, 141, 55, 218, 7, 35, 8, 176, 40, 58, 16, 104, 188, 109, 111, 192,
165, 132, 195, 24, 211, 94, 229, 98, 62, 6, 175, 11, 129, 245, 119, 18, 89, 95, 2, 123, 122, 32, 78, 237, 35, 1
22, 97, 129, 79, 84, 15, 130, 210, 137, 113, 236, 176, 91, 169, 249, 10, 7]}], 'reference': None}}], 'milestone'
: None, 'indexation': None, 'receipt': None, 'treasury_transaction': None}, 'nonce': 237994}
```

Prüfen Sie erneut den Kontostand:

Kontostand Anna: 9.0 MIOTA

Kontostand Bob: 1.0 MIOTA

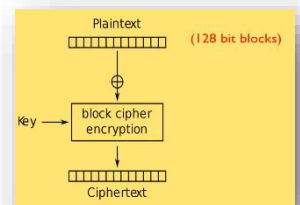
6 Verschlüsselung

In diesem Beispiel soll ein Text verschlüsselt werden. Hierfür benutzen wir das Verschlüsselungsverfahren Advanced Encryption Standard (AES).

6.1 Symmetrische Verschlüsselung mit AES

AES ist ein symmetrisches Verschlüsselungsverfahren, das einen gemeinsamen Schlüssel für die Verschlüsselung und Entschlüsselung von Daten verwendet. Es funktioniert folgendermaßen:

1. **Schlüsselaustausch:** Bevor Daten gesendet werden, müssen die beteiligten Parteien einen gemeinsamen Schlüssel vereinbaren. Dieser Schlüssel muss sicher übertragen werden, um sicherzustellen, dass keine unbefugte dritte Partei Zugang zu ihm erhält.
2. **Verschlüsselung:** Sobald der Schlüssel ausgetauscht wurde, kann die versendende Partei die Daten mithilfe des AES-Algorithmus und des Schlüssels verschlüsseln. Der AES-Algorithmus teilt die Daten in Blöcke (128 Bit) auf und verschlüsselt jeden Block mit dem Schlüssel.
3. **Übertragung:** Die verschlüsselten Daten werden über eine sichere Verbindung an die empfangende Partei gesendet.
4. **Entschlüsselung:** Die empfangende Partei kann die Daten mithilfe desselben AES-Algorithmus und des gleichen Schlüssels, den sie von der versendenden Partei erhalten hat, entschlüsseln.

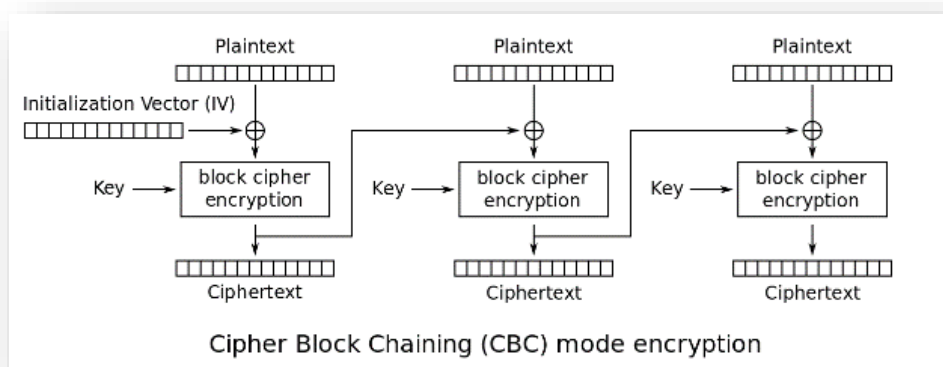


6.2 Passwörter Salzen

Da der AES-Algorithmus öffentlich bekannt ist, gibt es auch bekannte Angriffsszenarien, um das Passwort zu „knacken“. Wenn man z.B. den Kontext eines verschlüsselten Textes kennt und daher vermutet, dass ein bestimmtes Wort besonders häufig vorkommt (z.B. „BZTG“), kann man den verschlüsselten Text auf Wiederholungen analysieren und durch ausprobieren das Passwort ermitteln.

Um dies zu verhindern „salzt“ man die Passwörter. Für den ersten zu verschlüsselnden Block kombiniert man das Passwort mit einem zusätzlichen Wert, dem „Initialization Vektor“. Für alle weiteren Blöcke nimmt man als Salz das verschlüsselte Ergebnis des letzten Blocks.

So ergibt das gleiche Klartextwort (z.B. „BZTG“) mit dem gleichen Passwort, durch das Hinzufügen verschiedener „Salze“, niemals das gleiche Verschlüsselungsergebnis. Damit ist es unmöglich das Passwort zurückrechnen, indem man den verschlüsselten Text auf Wiederholungen analysiert.



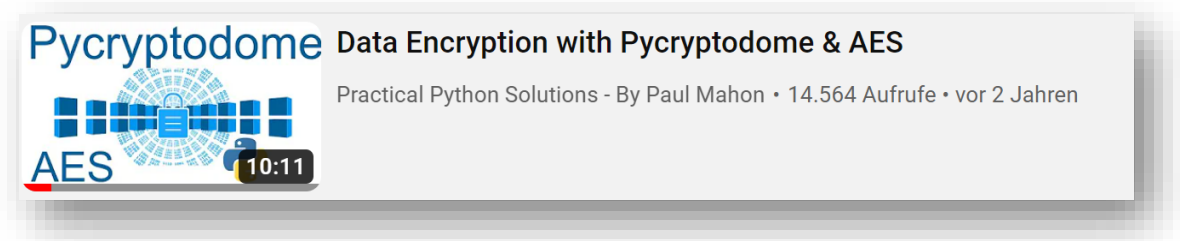
6.3 Padding

Padding bezieht sich auf den Prozess, bei dem Daten auf eine bestimmte Länge angepasst werden, um sie für eine sichere Verschlüsselung geeignet zu machen.

Im Kontext der AES-Verschlüsselung (Advanced Encryption Standard) müssen die Daten, die verschlüsselt werden sollen, in Blöcke von 128 Bit aufgeteilt werden, und es kann vorkommen, dass ein Block nicht die vollständige Größe hat. In diesem Fall müssen die Daten mit Nullen oder anderen Zeichen gefüllt werden, um die vollständige Blockgröße zu erreichen. Dies wird als „Padding“ bezeichnet.

Es ist wichtig zu beachten, dass das Padding nach der Verschlüsselung entfernt werden muss, um die ursprünglichen Daten wiederherzustellen. Dies nennt man „Unpadding“.

Zur Einführung können Sie folgendes Video ansehen: <https://www.youtube.com/watch?v=AUxHDIC6Rn0>



6.4 Die Bibliothek „pycryptodome“

Pycryptodome ist eine drittanbieter-Bibliothek für Kryptografie in Python. Es ist eine unabhängige Implementierung der Standardbibliothek Python Cryptography Toolkit (PyCrypto) mit aktualisierten Algorithmen und Sicherheitsfunktionen. Pycryptodome bietet eine Vielzahl von Kryptografie-Algorithmen wie AES, DES, RSA und viele andere, die zum Schutz von Daten und zur Authentifizierung von Benutzern verwendet werden können. Es ist ein sehr praktisches Tool für Entwickler, die Kryptografie in ihren Anwendungen einsetzen möchten.

Installieren Sie die Bibliothek „pycryptodome“: `py -m pip install pycryptodome`

6.5 Einen Text verschlüsseln

```
# Bibliotheken
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

# Initialisierung
key = b'mysecretpassword'          # 16 Byte Passwort
iv = b'password-salzen!'           # 16 Byte Initialization Vektor
cipher = AES.new(key, AES.MODE_CBC, iv) # Verschlüsselung initialisieren

# Verschlüsselung
plaintext = b'Die ist mein Klartext!' # Klartext
ciphertext = cipher.encrypt(pad(plaintext, AES.block_size)) # Text verschlüsseln

# Ausgabe
print ('-----')
print ("Verschlüsselter Text : ", ciphertext)
print ('-----')
```

Ausgabe:

```
-----
Verschlüsselter Text :  b'\xe0\xdc*\x84l\x87;p\xd2\xd9\x94\xabH6\xcd\xf0&\xedu0\x19\x17$+K*wke\x81\xdf'
-----
```

6.6 Einen verschlüsselten Text wieder entschlüsseln

Als verschlüsselter Text (ciphertext) wird die Ausgabe des vorherigen Programms verwendet.

```
# Bibliotheken
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

# Initialisierung
key = b'mysecretpassword'          # 16 Byte Passwort
iv  = b'passwort-salzen!'         # 16 Byte Initialization Vektor
cipher = AES.new(key, AES.MODE_CBC, iv) # Verschlüsselung initialisieren

# Entschlüsselung
ciphertext = b'\xe0\xdc*\x84l\x87;p\xd22\xd9\x94\xabH6\xcd\xfd&\xedu0\x19\x17$+K*wke\x81\xdf'
plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size) # Text entschlüsseln

# Ausgabe
print ('-----')
print ("Entschlüsselter Text als Bytewert: ", plaintext)
print ("Entschlüsselter Text als String:  ", plaintext.decode())
print ('-----')
```

Ausgabe:

```
-----
Entschlüsselter Text als Bytewert:  b'Die ist mein Klartext!'
Entschlüsselter Text als String:    Die ist mein Klartext!
-----
```

6.7 Eine verschlüsselte Nachricht auf den Iota-Tangle schreiben

Hier werden Daten im JSON-Format erzeugt, anschließend verschlüsselt und als Data-Message auf den Iota-Tangle geschrieben.

```
# Bibliotheken
import iota_client
import json
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from datetime import datetime

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe'

# Sensordaten als dict anlegen
payload = {
    'Sensor-ID': 'TS1',
    'Timestamp': datetime.now().strftime("%d.%m.%Y %H:%M:%S"),
    'Temperatur': 7.2
}

# dict als JSON-String ablegen
plaintext_str = json.dumps(payload)

# String in Byte umwandeln
plaintext_byte = plaintext_str.encode("utf8")

# Verschlüsselung initialisieren
key = b'mysecretpassword'          # 16 Byte Passwort
iv = b'passwort-salzen!'          # 16 Byte Initialization Vektor
cipher = AES.new(key, AES.MODE_CBC, iv)

# Bytes verschlüsseln
ciphertext_byte = cipher.encrypt(pad(plaintext_byte, AES.block_size))

# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Message auf den Tangle schreiben
message = client.message(index="BZTG-CIPHER1", data = ciphertext_byte)

# Message ausgeben
print(message)
```

Ausgabe:

```
{'message_id': '21388316028aaf6c19e005a59aadb6c116c40f38ad49361a3f9f2afe6f983b2d', 'network_id': 651478833251580
4015, 'parents': ['069ab0723dede4fc37d639ec3fb4dcd1c6ff44b1617be1ac072d079b43c65878', '18b7654d8cf22b5e820e80f86
478efef18ea5dea1f245e6e0c1d0898c6c54565', 'aca90ba19379a8974e3936627f842f2d94a9bdcc7b327d4d3983abef5a99945b', 'f
067c2de36ca8a080a7f144e52b6964d2ee7133d082fe034aad0adfe01578de8'], 'payload': {'transaction': None, 'milestone':
None, 'indexation': [{'index': '425a54472d434950484552', 'data': [122, 4, 199, 158, 48, 180, 190, 195, 61, 38,
23, 191, 201, 10, 184, 119, 47, 24, 65, 102, 7, 77, 205, 79, 68, 91, 79, 31, 174, 140, 88, 64, 119, 233, 40, 40,
162, 223, 255, 138, 2, 239, 79, 210, 227, 185, 250, 57, 12, 50, 198, 91, 45, 202, 64, 112, 181, 235, 145, 156,
180, 158, 95, 99, 255, 31, 245, 176, 101, 254, 141, 39, 13, 133, 52, 10, 27, 87, 95, 197]}], 'receipt': None, 't
reasury_transaction': None}, 'nonce': 9223372036854779140}
```

6.8 Eine verschlüsselte Nachricht vom Iota-Tangle lesen

Hier werden Daten vom Iota-Tangle gelesen und entschlüsselt.

```
# Bibliotheken
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import iota_client
import json

# Knoten aus dem Chrysalis-Devnet
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe'

# Client mit einem Knoten verbinden
client = iota_client.Client(nodes_name_password=[[node_url]])

# Messages zu einem Index suchen
mlist = client.find_messages(indexation_keys=["BZTG-CIPHER1"])

# Schleife über alle gefundenen Messages
for message in mlist:

    # Verschlüsselter Text als Integer-Array
    ciphertext_int = message['payload']['indexation'][0]['data']

    # Integer-Array in Bytes konvertieren
    ciphertext_byte = bytes(ciphertext_int)

    # Verschlüsselung initialisieren
    key = b'mysecretpassword'          # 16 Byte Passwort
    iv = b'passwort-salzen!'          # 16 Byte Initialization Vektor
    cipher = AES.new(key, AES.MODE_CBC, iv)

    # Bytes entschlüsseln
    plaintext_byte = unpad(cipher.decrypt(ciphertext_byte), AES.block_size)

    # Bytes in String konvertieren
    plaintext_str = plaintext_byte.decode("utf8")

    # JSON-String in Dictionary-Objekt konvertieren
    json_dict = json.loads(plaintext_str)

    # Alle Schlüssel und Schlüsselwerte des Dictionarys ausgeben
    for key in json_dict:
        print(key + ': ' + str(json_dict[key]))
        print('-----')
```

Ausgabe:

```
Sensor-ID: TS1
-----
Timestamp: 24.02.2023 20:16:53
-----
Temperatur: 7.2
-----
```

7 Projektstufe 1

7.1 Scenario

- Der Hersteller „Food Solution Hildesheim“ des Dönerspießes bietet seinen Endkunden eine zertifizierte Kühlkette für alle Produkte an. Diese kann vom Endkunden leicht über einen QR-Code abgefragt werden.
- Jeder Dönerspieß bekommt vom Logistikunternehmen eine eindeutige Identifikationsnummer als QR-Code.
- Jeder Dönerspieß durchläuft während des Transports verschiedene Stationen in der Kühlkette. Ein beispielhafter Transportweg könnte folgendermaßen aussehen:
 - Der **Kühltransporter** (KT) holt die Ware ab und transportiert sie zum
 - lokalen Kühlhaus im **Güterverteilzentrum** (GVZ)
 - Ein **Kühltransporter** (KT) bringt die Ware in ein
 - **Güterverteilzentrum** (GVZ) in der Nähe des Endkunden
 - Ein **Kühltransporter** (KT) bringt die Ware zum Endkunden.
- Der Endkunde kann die Einhaltung der Kühlkette überprüfen, indem er den QR-Code einscannt.

7.2 Technische Spezifikationen

- Die Zertifizierung der Kühlkette basiert auf den Informationen, die unveränderlich mit Hilfe von DLT abgespeichert werden.
- Jeder Kühlpunkt schreibt beim Ein- und Auschecken des Produktes folgende Daten auf den Tangle:

```
{'transportid':      '72359278599178561029671',  
'transportstation':  'GVZ-Hildesheim-Kühlhaus2-Zone4',  
'category':         'GVZ',  
'direction':        'in'/'out',  
'timestamp':        '05.09.2022 13:12:25'}
```

- Folgende Kriterien sind für die **Einhaltung der Kühlkette** zu überprüfen:
 - **Stimmigkeit der Kühlketteninformationen**
 - Gibt es für jede Transportstation jeweils einen Eintrag für das Ein-bzw. Auschecken?
 - Sind die Einträge zeitlich sinnvoll geordnet?
 - **Zeiträume ohne Kühlung**
 - Überschreitet die Zeit zwischen dem Auschecken aus einer Transportstation und dem Einchecken in die darauffolgende 10 min?
 - **Transportdauer**
 - Überschreitet die Gesamttransportdauer 48 h?

7.3 Aufgabenstellung

4 x 6er-Gruppe mit Partnerarbeit zu den drei Kriterien

Erstellen Sie ein Programm, dass die benötigten Daten anhand der übergebenen Transport-ID aus dem IOTA-Tangle ausliest, und die Einhaltung der drei Bedingungen „**Stimmigkeit der Kühlketteninformationen**“, „**Zeiträume ohne Kühlung**“ und „**Transportdauer**“ überprüft.

Das Programm soll dem Anwender die Information zurückgeben, ob alle Bedingungen an die **Kühlkette erfüllt** wurden oder nicht. Falls nicht, soll eine **eindeutige Fehlerbeschreibung** ausgegeben werden. Die Informationen sollen **übersichtlich** dargestellt werden.

Ermitteln Sie, mit Hilfe Ihres Programms, die Ergebnisse für den Index „**Food Solution Hildesheim**“ und die folgenden Transport-IDs:

- | | |
|-----------------------------|---|
| 1. 72359278599178561029675 | korrekt |
| 2. 15668407856331648336231 | Übergabe > 10 min |
| 3. 73491878556297128760578 | korrekt |
| 4. 99346757838434834886542 | Transportdauer > 48h |
| 5. 46204863139457546291334 | korrekt |
| 6. 77631003455214677542311 | korrekt |
| 7. 34778534098134729847267 | korrekt |
| 8. 64296734612883933474299 | Ausscheck-Zeitpunkt fehlt am Ende (kein Fehler, da Transport nicht abgeschlossen) |
| 9. 84356113249506843372979 | korrekt |
| 10. 23964376768701928340034 | Ausscheck-Zeitpunkt fehlt in der Mitte |
| 11. 55638471099438572108556 | Einchecken Kühllager liegt zeitlich vor Auschecken LKW |
| 12. 84552276793340958450995 | korrekt |
| 13. 96853785349211053482893 | korrekt |
| 14. 68345254400506854834562 | Aus und wieder Einchecken im gleichen Kühllager |
| 15. 67424886737245693583645 | Doppelter Ausscheck-Zeitpunkt (Abstand 12 min) |
| 16. 85746762813849598680239 | korrekt |
| 17. 56993454245564893300000 | Es gibt gar keinen Eintrag |
| 18. 95662334024905944384522 | korrekt |
| 19. 13456783852887496020345 | korrekt |
| 20. 76381745965049879836902 | korrekt |

Die Abgabe erfolgt in Form einer ausführlich **dokumentierten** und **kompilierbaren Python-Datei** im Moodle-Kurs.

8 Projektstufe 2

Der Hersteller „Food Solution Hildesheim“ möchte die Kühlkettenüberwachung aus Projektstufe 1 folgendermaßen ergänzen:

8.1 Temperaturüberwachung der Kühlstationen

Scenario:

Es hat sich herausgestellt, dass trotz der Kühlkettenüberwachung mehrfach minderwertige Ware ausgeliefert wurde, da die Kühltemperaturvorgaben auf dem Transport nicht eingehalten wurden. Zur Qualitätssicherung sollen zukünftig die Temperaturen der Kühllhäuser und der Transportfahrzeuge ausgewertet werden.

Aufgabe:

Überprüfen Sie die Einhaltung der Temperaturgrenzwerte in den Kühlstationen. Programmieren Sie dazu eine Funktion mit folgenden Übergabeparametern:

- Eingabeparameter: `transportstation` , `checkin-Zeit` , `checkout-Zeit`
- Rückgabeparameter: `Fehlermeldungstext`

Bauen Sie die Funktion so in ihr Hauptprogramm ein, dass Temperaturverletzungen angezeigt werden.

Technische Spezifikation:

- Jede Kühlstation (Güterverteilzentrum und Kühltransporter) schreibt im zeitlichen Abstand von 15 min Temperaturdaten auf den Tangle:

```
{'transportstation': 'GVZ-Hildesheim-Kühlhaus2-Zone4',  
  'category': 'GVZ',  
  'temperature': 4.3,  
  'timestamp': '05.09.2022 13:12:25'}
```

- Der **Index** für die Temperaturdaten ist jeweils die `'transportstation'`.
- Die **Kühltemperaturen** dürfen den Bereich zwischen +2°C und +4°C nicht verlassen.

8.2 Verschlüsselung der Daten auf dem Tangle

Scenario:

Es hat Beschwerden von Kunden gegeben, dass die Lieferdaten öffentlich einsehbar seien, da sie unverschlüsselt abgespeichert werden. Das hat die Firma „Food Solution Hildesheim“ inzwischen geändert und die Daten verschlüsselt auf den Tangle geschrieben.

Aufgabe:

Ändern Sie Ihr Programm aus Projektstufe 1 so ab, dass die verschlüsselten Daten aus dem Tangle verarbeitet werden können.

Technische Spezifikation:

Der Such-Index lautet „**Food Solution Hildesheim encrypted**“

Der Message-Payload ist folgendermaßen verschlüsselt:

- AES-CBC-Verschlüsselung mit Python-Pycryptodome
- Passwort: `b'mysecretpassword'`
- Initialization-Vector: `b'passwort-salzen! '`

8.3 Reparationszahlung bei Verletzung der Kühlkette

Scenario:

Das Logistikunternehmen garantiert der „Food Solution Hildesheim“ die Einhaltung der Kühlkette. Bei Verletzung der Kühlkette, soll eine Reparationszahlung geleistet werden. Dieser Vorgang soll programmgesteuerter über einen Smart-Contract automatisiert werden.

Aufgabe:

Programmieren sie eine Smart-Contract-Funktion, die bei einer Verletzung der Kühlkette automatisch „einen MIOTA“ an die „Food Solution Hildesheim“ überweist.

Bauen Sie die Funktion in Ihr Hauptprogramm ein.

Technische Spezifikation:

- Überweisungsbetrag bei Kühlkettenverletzung: 1 MIOTA
- Die Seeds und Adressen für die „Food-Solutions-GmbH“ und das Logistikunternehmen“ können selbst erzeugt und im Programmcode hinterlegt werden.

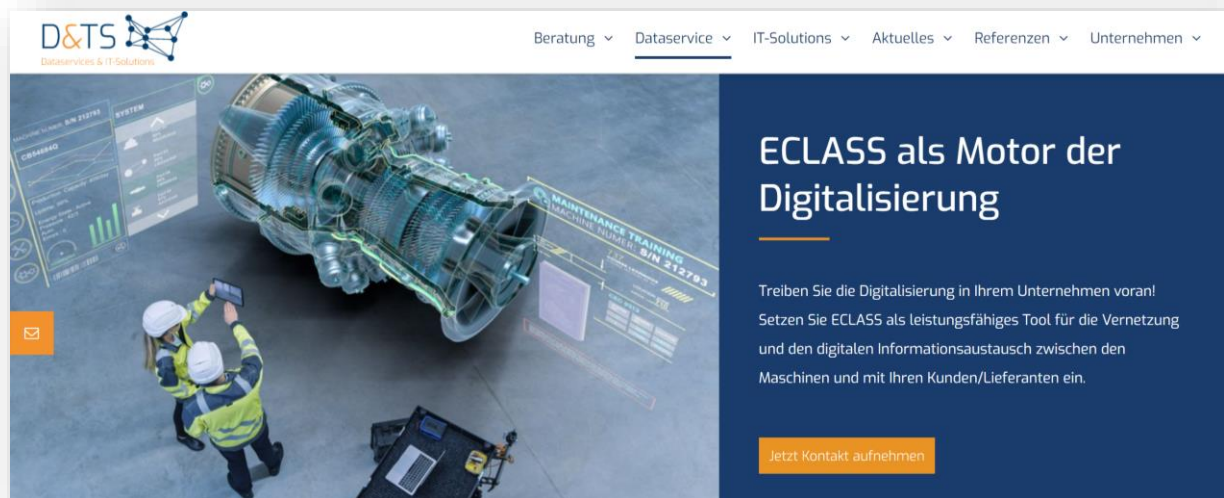
8.4 Abgabe

Alle Funktionen der Projektstufe 1 und 2 sollen in einer kommentierten Datei zusammengefasst werden. Diese Datei soll kompilier- und ausführbar sein. Das Programm soll die 20 verschlüsselten Transport-IDs auf Kühlkettenverletzung untersuchen und ggf. eine Fehlerbeschreibung ausgeben. Für jede Transport-ID mit Fehlern wird ein Reparationsbetrag überwiesen. Die Kontostände werden am Beginn und Ende des Programmdurchlaufs ausgegeben.

Geben Sie die Programmdatei über Moodle ab.

9 ECLASS

Beschreibung von digitalen Zwillingen



The screenshot shows the website of D&TS (Data Services & IT Solutions). The header includes navigation links: Beratung, Dataservice, IT-Solutions, Aktuelles, Referenzen, and Unternehmen. The main visual is a large industrial machine, possibly a turbine or engine, with a semi-transparent digital overlay showing internal components and data. Two workers in high-visibility gear and hard hats are in the foreground, one holding a tablet. On the left, a sidebar displays various data visualizations like bar charts and line graphs. On the right, a dark blue panel contains the text 'ECLASS als Motor der Digitalisierung' and a call to action 'Jetzt Kontakt aufnehmen'.

ECLASS als Motor der Digitalisierung

Treiben Sie die Digitalisierung in Ihrem Unternehmen voran!
Setzen Sie ECLASS als leistungsfähiges Tool für die Vernetzung und den digitalen Informationsaustausch zwischen den Maschinen und mit Ihren Kunden/Lieferanten ein.

[Jetzt Kontakt aufnehmen](#)

10 MQTT

10.1 Eine Nachricht mit Temperaturdaten per MQTT lesen

```
import iota_client
import json
import queue

# The node mqtt url
node_url = 'https://api.lb-0.h.chrysalis-devnet.iota.cafe/'

# The MQTT broker options
broker_options = {
    'automatic_disconnect': True,
    'timeout': 5,
    'use_ws': True,
    'port': 443,
    'max_reconnection_attempts': 5,
}

def on_mqtt_event(event):
    """Put the received event to queue.
    """
    q.put(event)

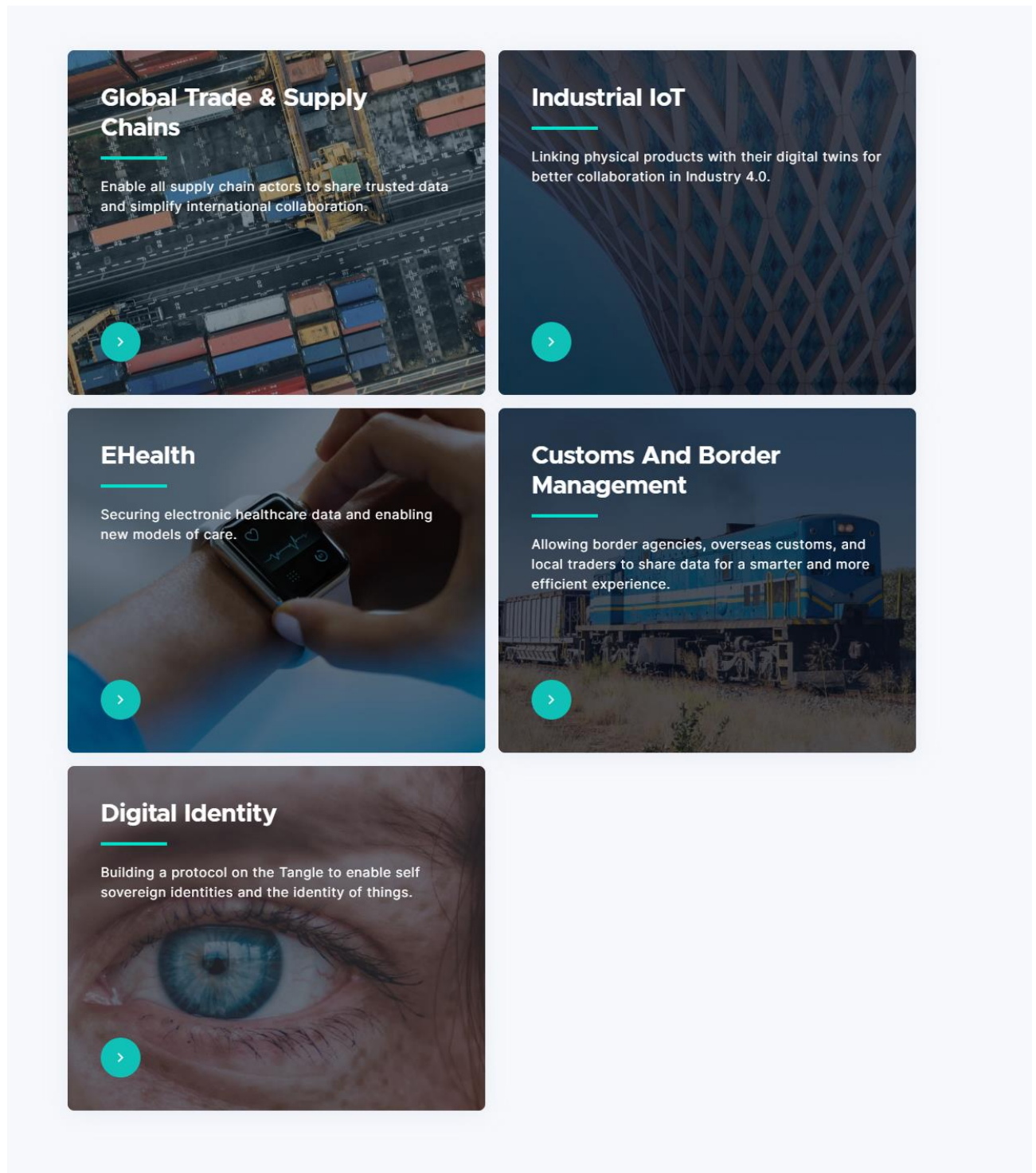
if __name__ == '__main__':
    client = iota_client.Client(nodes_name_password=[[node_url]], mqtt_broker_options=broker_options)
    client.subscribe_topics(['messages/indexation/RSE-TS1'], on_mqtt_event)

    received_events = 0
    q = queue.Queue()          # The queue to store received events
```

11 DLT-Anwendungsszenarien

Unter folgendem Link, werden erste reale Anwendungsszenarien für den IOTA-Tangle vorgestellt:

<https://www.iota.org/solutions/industries>



12 DLT-Detailwissen

12.1 Permissionless DLT

Ein DLT kann zwei Arten von Hauptbüchern haben:

Permissionless Ledger (dt. Zulassungsloses Kassenbuch): Ein Kassenbuch, das auf Nodes verteilt ist, die von jedermann ohne Erlaubnis ausgeführt werden können. Der Zweck eines zulassungslosen Kassenbuchs besteht darin, jedem zu ermöglichen, Daten zum Kassenbuch beizutragen, und für jedem, der das Kassenbuch besitzt, identische Kopien aller darin aufgeführten Transaktionen zu geben. Nodes erhalten die Integrität des Kassenbuchs aufrecht, indem ein Konsens über seinen Zustand erzielt wird. Ein zulassungsloses Ledger kann als unabänderlicher globaler Datensatz für Übertragungen verwendet werden.

Permissioned ledger (dt. Zugelassenes Kassenbuch): Ein Kassenbuch, das nur an Nodes verteilt wird, die von einer zentralen Behörde wie einer Bank oder einer Regierung vorgewählt werden.

12.2 Merkmale

Die wichtigsten Merkmale und Vorteile sind:

12.2.1 Dezentralität

Das Netzwerk wird von keiner zentralen Instanz kontrolliert, alle Netzwerkteilnehmer sind gleichberechtigt, daher kann das Netzwerk nicht von einem einzelnen Akteur abgeschaltet werden. Zudem besteht in einem dezentralen Netzwerk kein Single Point of Failure (dt. einzelner Ausfallpunkt) in dem technischen System, dessen Ausfall den Ausfall des gesamten Systems nach sich zieht, dies erhöht gleichzeitig auch die Sicherheit gegen potentielle Angreifer. Ein dezentrales Netzwerk ist sicherer gegen Manipulationen, es gibt eigene Validierungs- und Autorisierungsmechanismen quer durch das gesamte Netzwerk. Fälschungssichere mathematische Hash-Verfahren machen die Daten vertrauenswürdig. Die Integrität ist sichergestellt, da tausende von Nodes jede Transaktion validieren.

12.2.2 Unveränderlichkeit

Einmal durchgeführte Transaktionen können nicht verändert oder rückgängig gemacht werden. Bei der DLT gibt es nur eine einzige „Quelle der Wahrheit“. Jede unautorisierte Änderung im Netzwerk wird direkt offengelegt, sodass man sich der Richtigkeit der Transaktionen absolut sicher sein kann.

12.2.3 Programmiertes Vertrauen

Vertrauen, welches vormals von Vermittlern (Banken, Händler, etc.) als Dienstleistung angeboten wurde, kann nun technisch zur Verfügung gestellt werden. Seit der Finanzkrise 2008 hat das Vertrauen in die traditionellen Institutionen der Wirtschaft, des Finanzmarktes und sogar des Staates deutlich abgenommen. Mit der DLT können zwei Personen oder Unternehmen, die einander nicht kennen, miteinander ins Geschäft kommen ohne dem anderen vertrauen zu müssen und das auch noch ohne Vermittler. Das spart Zeit (keine Verträge etc.) und Geld (keine Vermittler Provisionen).

12.2.4 Doppelausgabenproblem ist gelöst

Einer der Hauptvorteile der dezentralen Technologien besteht darin, dass das Doppelausgabenproblem gelöst wird. Erklärung: Da digitales Geld nur eine Computerdatei ist, ist es einfach, durch einfaches Kopieren und Einfügen zu fälschen. Ohne DLT verfolgen die Banken das Geld aller Akteure auf ihren Konten, sodass niemand „doppelte Ausgaben“ bzw. das gleiche Geld zweimal ausgeben kann. Die DLT löst dieses Problem anders und effizienter als Banken: Es macht alle Transaktionen und Konten öffentlich, sodass es sofort offensichtlich ist, wenn Geld zweimal verwendet wird.

12.2.5 Transparenz

Jeder Nutzer kann alle Transaktionen nachverfolgen, es ist ein völlig transparentes System. Hinweis: Es gibt auch sogenannte „privacy“ Lösungen.

12.2.6 Sofortige Transaktionen

Transaktionen benötigen viel weniger Zeit als Transaktionen, bei denen eine Art Zwischenhändler (Bank) erforderlich ist.

Quelle: <https://iota-einsteiger-guide.de/grundlagen/archiv-distributed-ledger-technologie-dlt/>