Universal Credit System

Dokumentation

Willkommen!

In diesem Dokument finden Sie eine detaillierte technische Dokumentation des Universal Credit Systems. Es enthält alle Informationen die Sie benötigen um das Programm zu implementieren oder nach Ihren Bedürfnissen anzupassen.

Inhalt

1. Abhängigkeiten

Das Programm ist abhängig von anderen Programmen. Beim Setup überprüft das Script **`install.sh`** ob diese Programme installiert sind. Die folgenden Programme werden verwendet:

name	**verwendung**
awk	Sortieren/Filtern von Daten
basename	Trennen von Verzeichnissen und Suffix vom Dateinamen
bc	Gleitkommaberechnungen
cat	Dateien zusammenführen, Dateien ausgeben
chmod	Ändern von Zugriffsrechten
cp	Dateien kopieren
curl	Senden und Empfangen der TSA-Dateien
cut	Extrahieren von Strings
date	Datumsoperationen
dd	Dateikonvertierung
dialog	GUI
dirname	Trennen von Suffix vom Dateinamen
echo	Ausgabe Bildschirm/Datei
expr	Berechnungen
find	Suchen von Dateien/Verzeichnissen
flock	Verwaltung von Dateisperrungen für multi user setups
gpg	Signieren
grep	Dateien durchsuchen
head	Ausgabe der ersten Zeilen/Zeichen
ls	Auflisten von Dateien und Verzeichnissen
mkdir	Erstellen von Ordnern und Unterordnern
mv	Verschieben von Dateien
netcat	Senden/Abfragen von Daten bei UCA
openssl	Verifizieren von TSA-Dateien
printf	Ausgabe Bildschirm/Datei
rm	Löschen von Dateien
sed	Lesen und Verändern von Text
sha224sum	Hash-Funktion
sha256sum	Hash-Funktion
sort	Listen sortieren
stat	Ermitteln von Datei-/Verzeichnis-Rechten
tail	Ausgabe der letzten Zeile
tar	Erstellen von Archiven
test	Testen von Dateien
touch	Initiale Erstellung von Dateien
tr	Umwandlung von Zeichen
umask	Ermittlung der user-mask
uniq	Listen filtern
wc	Zählen von Zeilen, Zeichen und Bytes
wget	Download von Zertifikatsdateien

2. Ordner

Name	Beschreibung
~/assets/	**enthält die Assets der Benutzer**
~/backup/	**enthält die Backups der Benutzer**
~/certs/	**enthält die TSA-Zertifikate**
~/control/	**enthält wichtige Systemdateien**
~/keys/	**enthält die öffentlichen Schlüssel der Benutzer**
~/lang/	**enthält die Sprach-Dateien**
~/proofs/	**enthält die proofs der Benutzer**
~/theme/	**enthält die Themes für das GUI**
~/trx/	**enthält die Trx der Benutzer**
~/userdata/	**enthält temporäre und Benutzerspezifische Dateien**

3. Details

~/assets/

Enthält die von Benutzer erstellten Assets. Assets sind im Grunde genommen Tokens. Die Namenskonvention für Asset-Dateien ist wie folgt:

<ASSET_SYMBOL**>**.<ZEITSTEMPEL>

<*ASSET_SYMBOL*> kann ein bis zu 10 Zeichen langer String sein.
<*ZEITSTEMPEL*> ist der Zeitpunkt der Erstellung im Unix Timestamp Format (unix epoch time).

~/backup/

Enthält System Backups, die der Benutzer erstellt hat. Mit Hilfe dieser Backups ist es möglich die Daten wiederherzustellen, falls sie korrupt sind. Die Namenskonvention für Backup-Dateien ist wie folgt:

<ZEITSTEMPEL**>**.bc

<*ZEITSTEMPEL*> ist die Zeit im Unix Timestamp Format (unix epoch time).

~/certs/

Dieser Ordner enthält einen Unterordner für jede TSA. In diesem Unterordner befinden sich die initialen Zertifikate. Aktuell wird lediglich freeTSA als TSA akzeptiert.

Folgende Dateien befinden sich im Unterordner:

cacert.pem (Zertifikat der CA)
tsa.pem (Zertifikat der TSA)

Diese Zertifikate werden verwendet um die TSA-Dateien im ~/**proofs/** Verzeichnis des Benutzers mit ***openssl ts*** zu überprüfen.

~/control/

Dieser Ordner enthält wichtige Systemdateien, welche von UCS verwendet werden:

config.conf	Konfigurations-Datei
install_config.conf	default Konfigurations-Datei
tsa.conf	TSA Liste
uca.conf	UCA Liste
dh.db	Datenbank für Perfect Forward Secrecy
install.dep	Liste mit den abhängigen Paketen
keyring.file	GPG-Keying
HELP.txt	Hilfetext

Des weitere werden alle privaten Schlüssel von Benutzern, welche auf diesem Computer erstellt werden, im Ordner ***~/control/keys/*** gespeichert.

~/keys/

Dieser Ordner enthält die öffentlichen Schlüssel der Benutzer. Die Schlüssel sind 4096Bit RSA-Schlüssel, welche mit GnuPG erstellt und in der Datei ***keyring.file*** im ***~/control/*-Ordner** verwaltet werden. Die Namenskonvention ist wie folgt:

<ADRESSE>.<ZEITSTEMPEL>

<ADRESSE> ist ein SHA-256 hash, welcher berechnet wird indem die Zeichenkette „*<ACCOUNTNAME>*_*<PIN>*_*<ZEITSTEMPEL>*“ gehasht wird. *<ZEITSTEMPEL>* ist der Zeitpunkt der Erstellung im UNIX timestamp Format (unix epoch time). Dieses Verfahren wird auch beim Logon angewendet; der Benutzername und die PIN werden zusammen mit den Zeitstempeln gehasht um den passenden Schlüssel zu finden.

~/lang/

Dieser Ordner enthält die Sprach-Dateien, welche vom Script automatisch importiert werden und dann im GUI zur Auswahl stehen. Die gewählte Sprachdatei wird beim start von ucs_client.sh gesourced. Die Namenskonvention ist wie folgt:

lang_<Ländercode>_<Sprache-lang>.conf

<Ländercode> ist der Code des Landes z.B. ***EN*** für ***ENGLAND*** or ***FR*** for ***FRANCE***. *<Sprache-lang>* ist der Name der Sprache in der jeweiligen Sprache z.B. ***SVENSKA*** für s*chwedisch* oder ***ITALIANO*** für *italienisch*. Sie können neue Dateien hinzufügen und das Script wird diese automatisch zu der Liste der verfügbaren Sprachen hinzufügen.

Ändern/löschen Sie beim übersetzen jedoch auf keinen Fall die Tags '<tag>' oder '/n'! Diese Tags dienen der Formatierung für die Darstellung der grafischen Benutzeroberfläche mit Hilfe von `dialog`!

Alles andere kann ensprechend einer anderen Sprache geändert werden. Wenn die Sprach-Dateien ensprechend der Namenskonvention benannt sind erkennt das Script die Dateien und Sie können sie im GUI auswählen

~/proofs/

Dieser Ordner enthält für jeden Benutzer einen Unterordner und in diesem Unterordner befinden sich die *proofs* (index-Datei, TSA Anfrage, TSA Antwort) des Benutzers. Die Unterordner haben folgende Namenskonvention:

/<ADRESSE>.<ZEITSTEMPEL>/

<ADRESSE>.<ZEITSTEMPEL> entsprechen dem Benutzernamen.

Die TSA-Dateien im proofs/-Ordner werden mit Hilfe des Kommandos **openssl ts** überprüft.

Folgende Dateien befinden sich im proofs/-Ordner eines Benutzers:

freetsa.tsq	(freeTSA Anfrage)
freetsa.tsr	(freeTSA Anwort)
<ADRESSE>.<ZEITSTEMPEL>.txt	(Index-Datei)

Die Index-Datei enthält eine Liste aller vom Benutzer annerkannten Benutzer inklusive deren proofs sowie eine Liste aller annerkannter Transaktionen. Diese Datei wird mit Hilfe von GnuPG verifiziert.

~/theme/

Dieser Ordner enthält die Themes, welche von dialog für das GUI verwendet werden. Jedes Theme, welche in diesem Ordner abgelegt wird, wird automatisch vom Script erkannt und steht dann unter Hauptmenü → Einstellungen → Themes zur Auswahl.

~/trx/

Dieser Ordner enthält die Transaktionen aller Benutzer. Die Namenskonvention ist wie folgt:

<ADRESSE>.<USER_ZEITSTEMPEL>.<TRX_ZEITSTEMPEL>

<ADRESSE>.<USER_ZEITSTEMPEL> ist die Adresse des Senders während *<TRX_ZEITSTEMPEL>* der Erstellungszeitpunkt der Transaktion im UNIX timestamp format (unix epoch time) ist. Transaktionsdateien sind simple Textdateien.

~/userdata/

Enthält einen Unterordner für jeden Benutzer, der sich auf diesen Computer eingeloggt hat. Innerhalb dieses Unterordners befinden sich Benutzerspezifische Dateien:

all_assets.dat	Liste aller Assets
all_keys.dat	Liste aller Schlüssel
all_accounts.dat	Liste aller Accounts
all_trx.dat	Liste aller Transaktionen
blacklisted_accounts.dat	Liste aller gelöschter Benutzer
blacklisted_trx.dat	Liste aller gelöschter Transaktionen
depend_accounts.dat	Liste aller abhängigen Benutzer
depend_trx.dat	Liste aller abhängigen Transaktionen
depend_confirmations.dat	Liste aller abhängigen Transaktionen mit zu wenig Bestätigungen
<YYYYMMDD>_ledger.dat	Ledger-Datei des jeweiligen Datums
<YYYYMMDD>_scoretable.dat	Score-Datei des jeweiligen Datums
<YYYYMMDD>_index_trx.dat	Liste von Transaktionen, welcher der Benutzer bestätigt hat am jeweiligen Datum

Des weiteren werden Dateien, welche aus Transaktionen oder Synchronisations-Dateien entpackt werden, unter ***~/userdata/<Benutzername>/temp/*** gespeichert. Diese Dateien werden zunächst in den /temp Ordner des Benutzers entpackt und von dort entweder verschoben oder gelöscht.

4. Beispiel Transaktion

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

:TIME:1665066890
:AMNT:1.000000000|UCC
:SNDR:9d8c98a97b2c3e689afef90310a35130bde86fd6f43ef6764b391c40ba37f8dd.1613477808
:RCVR:ca2c6f1d030c0ea7e56893a89c32d6c86478b56ff40cfb327608ef47a58bc401.1613477644
:PRPS:86954a568d85d4e6b73569f5f8b2c44a956ba5b7acfcf55fca29159c
-----BEGIN PGP SIGNATURE-----

iQIzBAEBCgAdFiEEDWT9IMjKKYd6rKhIT+61NsV5pQUFAmM+54sACgkQT+61NsV5
pQXtXxAAwlE6Eift7OA+k+3CC0lzeBRjpqRoJmMKkeyH9dXjyHf2nEThx3mFADJ0
mppAPkrCHh9qGicsbv4xwU9ciXLClNgNu3RaZ3GrHPKfJR3FUTy/xKVRSaw/+S19
votpKy0HlLb6Lke9gLDjOhVLbQ0/EqwGfbLT2/mOiAavdKKmpDBc8QBqWyfFY8+B
pQNDXXESz68FX2hboFOvRMvyF0QRzFMNrpsjkG4W1hiSAoJ5ljKBokneDDsnV5FC
I887ds8gKu2sEpTYxwy2jI4M5oa39g6gtk/bJh/b3HxTk3QNULkIwrvOGes3KXfi
6ik8X2p37yeBD/HYNpH2c5ViHqAvjlFdwemShZWSX9UXTKGHBry2dNROx/yPzib4
rsHFKN/2l5UQ/0OwA0V8s8eEDpIYbrbQ/KqSLrXW45jGwI/w/tWJe4gqRPM27JAg
PYic+FwLUnYeEF93RIKsPB/Wtoa4vCsO8pPjvgVDav8WQh9YmyPmsU4AjMo7i4Jl
4hHg+R1ttRbXCBz9PQ+TxQHRFfOFaJRfEifHj2Aqli4PCxuWLx047UTc6PxlBE0w
cwHgz3Of0suc5DEyXhDD+eBSMUzNS4/NMlT30VvOauoMnoWDrTFgTddqlnX65T7O
V1Vkf3t5n99ys/wz9qRnRzFAHIAN0yfYfdApqO7K4+RUXktQW1s=
=g2B2
-----END PGP SIGNATURE-----
```

Eine Transaktion ist eigentlich nichts anderes als ein lesbar signierter Text im OpenPGP Format. Sie enthält alle nötigen Informationen: das *Datum der Erstellung* (TIME), den *zu überweisenden Betrag* mit Asset-Definition (AMNT), den *Absender der Transaktion* (SNDR), den *Empfänger der Transaktion* (RCVR) und einen *Verwendungszweck* (PRPS).

5. Wallet Installation

In der Annahme, dass Sie APT als packaging tool verwenden wird das Kommando **`apt-get install`** verwendet. Wenn Sie ein anderes packaging tool als APT verwenden müssen Sie das Kommando entsprechend anpassen. Ändern Sie in diesem Fall **`apt-get install`** ab!

Git installieren (eventuell **`sudo`** verwenden):

```
apt-get install git
```

Erstellen Sie ein Verzeichnis und wechseln Sie in dieses Verzeichnis:

```
mkdir ucs
cd ucs
```

Klonen Sie das GitHub repository und welchseln Sie in dieses Verzeichnis:

```
git clone https://github.com/universal-credit-system/wallet
cd wallet/
```

Nun können Sie das Script **`install.sh`** ausführen. Das Script überprüft zunächst ob alle notwendigen Programme installiert sind. Falls ja wird das Setup durchgeführt. Falls ein benötigtes Programm noch nicht installiert ist wird das Script den Namen des oder der Programme ausgeben und sich beenden. In diesem Fall müssen Sie erst die notwendigen Programme installieren und danach das Script **`install.sh`** noch einmal ausführen:

```
./install.sh
```

Nach dem Setup können Sie **`ucs_client.sh`** starten:

```
./ucs_client.sh
```

6. Assets

Das Universal Credit System unterstützt die Einrichtung von Tokens, hier generell als 'Assets' bezeichnet. Diese *Assets* können wechselbar sein (fungible) oder nicht wechselbar sein (non-fungible). Diese Assets werden im Ordner **~/assets/** gespeichert. Der Hauptunterschied von Assets zur Währung UCC ist, dass beim Transfer von Assets **kein Scoring** angewandt wird.

FUNGIBLE ASSETS

Fungible assets sind von allen Benutzern konvertierbarer Werte. Das bedeutet, dass jeder Nutzer sein Guthaben in diese Werte umwandeln kann. Der Umtausch von *fungible assets* ist nicht limitiert; es gibt keine maximale Menge dieser Assets. Hier ist ein Beispiel für die Definition eines fiktiven assets 'TestFungibleToken' mit dem Asset-Symbol 'TFT.1655676000':

Beispiel fungible asset '**TFT.1655676000':**

```
asset_description=TestFungibleToken
asset_price=2.000000000
asset_fungible=1
```

asset_description ist die Beschreibung des Assets, *asset_price* ist der Preis pro Einheit in UCC. *asset_fungible=1* gibt an, dass es sich um ein konvertierbares Asset handelt. Fungible assets von anderen Benutzern werden **nicht** automatisch importiert. Hierfür muss in der Datei **control/config.conf** der Wert **'import_fungible_assets=1'** gesetzt werden.

NON-FUNGIBLE ASSETS

Non-fungible assets sind nicht konvertierbarer Werte. Nicht konvertierbar bedeutet, dass man ein solches Asset entweder überwiesen bekommen muss oder man muss selbst der initiale Besitzer sein. Die totale Anzahl eines non-fungible assets ist limitiert auf den als **asset_quantity** festgelegten Wert. Hier ein Beispiel für die Definition eines fiktiven Assets 'TestNonFungibleToken' mit dem Asset-Symbol 'TNFT.1655676000':

Beispiel non-fungible asset '**TNFT.1655676000':**

```
asset_description=TestNonFungibleToken
asset_owner=9d8c98a97b2c3e689afef90310a35130bde86fd6f43ef6764b391c
40ba37f8dd.1613477808
asset_quantity=100.000000000
asset_fungible=0
```

asset_description ist die Beschreibung des Assets, *asset_owner* ist der initiale Besitzer, *asset_quantity* ist die Anzahl an assets. *asset_fungible=0* gibt an, dass es sich um ein nicht konvertierbares Asset handelt. Non-fungible Assets von anderen Benutzern werden **nicht** automatisch importiert. Hierfür muss in der Datei **control/config.conf** der Wert **'import_non_fungible_assets=1'** gesetzt werden.

7. Wie man eine DAO erstellt

In diesem Thema geht es darum, wie man eine *dezentrale autonome Organisation* (DAO) erstellt.

Um zu verstehen, wie in UCS die Idee der DAOs umgesetzt wurde, stellen Sie sich einfach die Aktien eines Unternehmens vor: je nachdem, wie viele Aktien Sie besitzen, besitzen Sie auch einen Teil dieses Unternehmens. Irgendwann werden die Gewinne dieses Unternehmens unter den Eigentümern der Aktien verteilt. Der Prozentsatz der Gewinne, die Sie erhalten, entspricht dem Prozentsatz des Unternehmens, das Sie durch die Aktien besitzen. In UCS beschränkt sich das Konzept der DAOs auf die faire Verteilung der an diese DAO gesendeten Beträge unter den DAO-Teilnehmern.

Aus technischer Sicht besteht eine DAO aus zwei Assets: einem fungible asset, welches verwendet wird, um die Einnahmen zu erhalten, und einem fungible oder non-fungiblee asset, welches dazu verwendet wird, um die Eigentümsverhältnisse der DAO widerzuspiegeln. Die „Aktien“ sind also einfach fungible oder non-fungible assets! Jeder Eigentümer eines solchen Assets wird als „Aktieninhaber“ betrachtet. Das Asset, welches verwendet wird, um diese Eigentumsrechte widerzuspiegeln, wird als „asset_owner=“ des Assets definiert, welches verwendet wird, um die Einnahmen zu erhalten. Das erste Asset fungiert einfach ausgedrückt als zentrale Empfängeradresse für die Eigentümer des zweiten Assets.

Die DAO hat keinen Abstimmungsprozess darüber, was mit dem Guthaben geschehen soll, wie es zum Beispiel innerhalb der Ethereum DAO der Fall ist. Die Nutzer müssen Vereinbarungen außerhalb von UCS treffen und ihre Salden dann selbst an eine zweite DAO weiterleiten, die die spezifische Investition abwickelt.

SCHRITT 1 : ERSELLEN SIE EIN FUNGIBLE ASSET

Dieses Asset dient als Adresse der Besitzer des Assets DAO.1655676000.

Beispiel fungible asset '**DAOT.1655676000':**

```
asset_description=DAOToken
asset_owner=DAO.1655676000
asset_price=1.000000000
asset_fungible=1
```

SCHRITT 2 : ERSELLEN SIE EIN FUNGIBLE ODER NON FUNGIBLE ASSET

Wählen Sie das *non-fungible asset* wenn sie eine feste Anzahl ein Besitzrechten festlegen möchten, welche nicht erhöht werden kann. Das bedeutet, dass andere Benutzer der DAO nicht eigenständig beitreten können.

Beispiel non-fungible asset '**DAO.1655676000**':

```
asset_description=DAO
asset_quantity=1000.000000000
asset_owner=9d8c98a97b2c3e689afef90310a35130bde86fd6f43ef6764b391c40ba37f
8dd.1613477808
asset_fungible=0
```

Wählen Sie das *fungible asset* wenn sie keine feste Anzahl ein Besitzrechten festlegen möchten. Das bedeutet, dass andere Benutzer der DAO beitreten können indem sie Geld an dieses Asset senden.

Beispiel fungible asset '**DAO.1655676000**':

```
asset_description=DAO
asset_price=1.000000000
asset_fungible=1
```

8. CMD-Modus

Im Folgenden finden Sie detaillierte Beispiele über die Kommandos des CMD-Modus. Mit diesen Kommandos ist es z.B. sehr einfach eine automatisierte Zahlungsabwicklung zu realisieren.

WIE MAN EINEN BENUTZER ERSTELLT

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action create_user -user TESTUSER -password
TESTPASSWORD

AUSGABE:
USER:<ACCOUNTNAME>
PIN:<PIN>
PASSWORD:>PW<          # HINWEIS: PASSWORD WIRD IN >< AUSGEGEBEN
ADRESS:<ADRESSE>
KEY:<KEYFILE>
KEY_PUB:/keys/<ADDRESSE>.<ZEITSTEMPEL>
KEY_PRV:/control/keys/<ADDRESSE>.<ZEITSTEMPEL>
```

HINWEIS: Zur Zeit werden die privaten Schlüssel immer im control/keys/-Verzeichnis und die öffentlichen Schlüssel immer im keys/-Verzeichnis abgelegt. Dies ist immer so und kann nicht geändert werden. Ein angegebener Pfad bleibt unberücksichtigt. Passen Sie gut auf die Schlüssel auf und machen Sie sicherheitshalber ein Backup.

WIE MAN EINE KLEINE TRANSAKTION ERSTELLT (falls möglich nur neue Dateien hinzufügen)

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action create_trx -user TESTUSER -pin 12345
-password TESTPASSWORD -receiver ADRESS -amount 1.000000000 -asset
ASSET -purpose "PURPOSE TEXT" -type partial -path
/path/to/outputdir
```

HIWEIS: Typ „partial“ bedeutet, dass das Programm überprüft ob Sender und Empfänger ein gemeinsames Transaktionswissen haben. Ist dies der Fall werden nur Dateien gepackt, die der Empfänger noch nicht besitzt. Dies kann die Größe einer Transaktionsdatei verringern.

WIE MAN EINE GROßE TRANSAKTION ERSTELLT (alle Dateien hinzufügen)

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action create_trx -user TESTUSER -pin 12345
-password TESTPASSWORD -receiver ADRESS -amount 1.000000000 -asset
ASSET -purpose "PURPOSE TEXT" -type full -path /path/to/outputdir
```

HINWEIS: Typ „full“ bedeutet alle Daten werden der Transaktionsdatei hinzugefügt unabhängig von gemeinsamen Transaktionswissen.

WIE MAN EINE TRANSAKTIONSDATEI PARTIELL LIESST (nur neue Dateien entpacken)

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action read_trx -user TESTUSER -pin 12345
-password TESTPASSWORD -type partial -path /path/to/file/file.trx
```

HINWEIS: Typ „partial“ bedeutet das Programm prüft das Gemeinsame Transaktionswissen und entpackt nur Dateien, welche der Sender noch nicht kennt. Dies ist der Standard und Sie sollten dies immer so machen! Damit verhindern Sie, dass die Dateien, welche Sie bereits besitzen überschrieben werden.

WIE MAN EINE TRANSAKTIONSDATEI KOMPLETT LIESST (alle Dateien entpacken)

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action read_sync -user TESTUSER -pin 12345
-password TESTPASSWORD -type full -path /path/to/file/file.trx
```

HINWEIS: Typ „full“ bedeutet, dass das Programm alle Dateien der Transaktionsdatei entpackt. Dies überschreibt die existierenden Daten und sollte nur mit höchster Vorsicht und Situationsbewusstsein gemacht werden. Dies erlaubt es Ihnen z.B. Ihre Daten nur auf Basis einer Transaktionsdatei wiederherzustellen falls sie defekt/korrupt sind.

WIE MAN EINE SYNCHRONISATIONS-DATEI ERSTELLT (enthält alle Dateien):

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action create_sync -user TESTUSER -pin 12345
-password TESTPASSWORD -path /path/to/outputdir
```

HINWEIS: Da eine Synchronisations-Datei keinen expliziten Empfänger hat enthält sie immer alle Dateien, welche im Besitz des sendenden Benutzers waren. Es liegt am Empfänger zu entscheiden, welche Dateien er entpackt (full oder partial).

WIE MAN EINE SYNCHRONISATIONS-DATEI PARTIELL LIESST (nur neue Dateien entpacken):

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action read_sync -user TESTUSER -pin 12345
-password TESTPASSWORD -type partial -path /path/to/file/file.sync
```

HINWEIS: Typ „partial“ bedeutet das Programm prüft das Gemeinsame Transaktionswissen und entpackt nur Dateien, welche der Sender noch nicht kennt. Dies ist der Standard und Sie sollten dies immer so machen! Damit verhindern Sie, dass die Dateien, welche Sie bereits besitzen überschrieben werden.

WIE MAN EINE SYNCHRONISATIONS-DATEI KOMPLETT LIESST (alle Dateien entpacken):

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action read_sync -user TESTUSER -pin 12345
-password TESTPASSWORD -type full -path /path/to/file/file.sync
```

HINWEIS: Typ „full“ bedeutet, dass das Programm alle Dateien der Synchronisationsdatei entpackt. Dies überschreibt die existierenden Daten und sollte nur mit höchster Vorsicht und Situationsbewusstsein gemacht werden. Dies erlaubt es Ihnen z.B. Ihre Daten nur auf Basis einer Transaktionsdatei wiederherzustellen falls sie defekt/korrupt sind.

WIE MAN SICH MIT EINER UCA SYNCRONISIERT

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action sync_uca -user TESTUSER -pin 12345
-password TESTPASSWORD
```

HINWEIS: Die Aktion “sync_uca” generiert keine Ausgabe wenn sie erfolgreich verlaufen ist. Des weiteren beendet sich das Programm immer mit Exit-Code 0. Der Exit-Code ist selbst dann '0', wenn das Senden/Empfangen zu/von einer oder mehrer UCAs fehlgeschlagen ist. Falls das Senden/Empfangen von Daten an eine oder mehere UCAs fehlgeschlagen ist gibt das Programm eine „ERROR“-Nachricht aus, welche die IP des UCA-Servers (<uca_ip>) und den verwendeten Port (<uca_snd_port>) beinhaltet. Dabei entsprechen <uca_ip> und <uca_snd_port> den Einträgen in *~/control/uca.conf*

WIE MAN EIN BACKUP ERSTELLT

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action create_backup
```

WIE MAN EIN BACKUP WIEDERHERSTELLT

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action restore_backup -path
/path/to/ucs/backup/<ZEITSTEMPEL>.bcp
```

WIE MAN DIE STATISTIKEN ANZEIGT

```
BEISPIEL KOMMANDO:
./ucs_client.sh -action show_stats -user TESTUSER -pin 12345
-password TESTPASSWORD
```

9. Universal Credit Contractor

Der Universal Credit Contractor fungiert als Wrapper-Script für den **`ucs_client.sh`** und ermöglicht dem Benutzer Smart Contracts für Transaktionen einzurichten. Wenn das bash Script **`ucs_contractor.sh`** ausgeführt wird, wird die entsprechende Logik gesourced und basierend auf dem ruleset die entsprechenden Aktionen ausgeführt.

WAS IST EIN CONTRACT?

Ein Contract besteht immer aus **mindestens** zwei Dateien:

- einer Datei im Ordner **`/contracts/`**, welche die Logik als Code enthält (genauer gesagt die Definition einer Funktion namens **`contract_action()`**, welche gesourced wird)
- einer Datei im Ordner **`/rulesets/`**, welche das Regelwerk enthält (Definitionen von Variablen mit denen die Logik gesteuert wird)

Beide Dateien werden dem **`ucs_contractor.sh`** per Parameter übergeben. Bei der Ausführung wird die Logik des Contracts geladen und mit den Variablen der Rulesets-Datei gesteuert. Grundsätzlich kann jede Logik realisiert werden.

INSTALLATION

SCHRITT 1: LADEN SIE DEN QUELLCODE RUNTER

Um den Contractor auszuführen, müssen Sie einen vollständigen Client mit Benutzer einrichten. Angenommen, Sie haben den Client bereits installiert, gehen Sie in das Verzeichnis und entpacken Sie den Contractor:

```
tar -xvf contractor.tar
```

Der Tarball enthält folgende Dateien/Ordner:

- das Script **`ucs_contractor.sh`**
- den Ordner **`/contracts/`**
- den Ordner **`/rulesets/`**
- die Datei /**`control/contractor_HELP.txt`**

Des Weiteren sind einige Beispiel-Contracts (cashier, filter, accountant und tombola) und die entsprechenden Rulesets im tarball enthalten.

SCHRITT 2: DEFINIEREN SIE EINEN SMART-CONTRACT

Erstellen Sie eine Smart-Contract-Logik und ein Smart-Contract-Ruleset nach Ihren Bedürfnissen.

BEISPIEL

Folgendes Beispiel ist ein Ruleset für den mitgelieferten Smart-Contract **`accountant.logic`**. Der Smart-Contract **`accountant.logic`** agiert als einfacher Buchhalter, der Transaktionen basierend auf empfangenen Transaktionen sendet.

Es können nur Parameter bezüglich der Transaktion als Trigger definiert werden und die Aktion beschränkt sich auf das Erstellen von neuen Transaktion(en).

Siehe folgendes Beispiel **accountant.ruleset**:

```
ruleset_asset="PUT_YOUR_ASSET_HERE"
ruleset_sender="*"
ruleset_receiver="PUT_YOUR_ADRESS_HERE"
ruleset_amount="*"
ruleset_amount_comparison_operator=""
ruleset_amount_comparison_variable=""
ruleset_purpose="*"
ruleset_required_confirmations=0
contract_sender="PUT_YOUR_ADRESS_HERE"
contract_sender_password="PUT_YOUR_PW_HERE"
contract_receiver="${trx_sender}"
contract_amount="${trx_amount}"
contract_purpose=`echo "${trx_file}"|sha256sum|cut -d ' ' -f1`
contract_type="partial"
alias send_trx='${script_path}/ucs_client.sh -action create_trx
-sender ${contract_sender} -password "${contract_sender_password}"
-receiver ${receiver} -amount ${contract_amount} -asset $
{contract_asset} -purpose ${contract_purpose} -type $
{contract_type}'
```

Das obige Ruleset sorgt dafür, dass der Smart-Contract **accountant.logic** alle Transaktionen, die an Sie gesendet wurden, an den Absender zurücksendet (wenn Sie das Asset, Ihre Adresse und Ihr Passwort eintragen).

accountant.logic sucht nach Transaktionen:

- die dem definierten Asset entsprechen (**ruleset_asset="PUT_YOUR_ASSET_HERE"**)
- die einen beliebigen Absender haben (**ruleset_sender="*"**)
- die Sie als Empfänger haben (**ruleset_receiver="PUT_YOUR_ADRESS_HERE"**)
- die einen beliebigen Betrag haben (**ruleset_amount="*"**)
- die einen beliebigen Überweisungszweck haben (**ruleset_purpose="*"**)
- die keine Bestätigungen benötigt (**ruleset_required_confirmations=0**)

Wenn eine Transaktion diesen Kriterien entspricht, erstellt der Contractor eine Transaktion:

- mit dem ursprünglichen gesendeten Asset als zu sendenden Asset (**contract_asset="${trx_asset}"**)
- mit dem ursprünglichen gesendeten Betrag als zu sendenden Betrag (**contract_amount="${trx_amount}"**)
- dem ursprünglichen Sender als Empfänger (**contract_receiver="${trx_sender}"**)
- mit einem sha256-Hash des ursprünglichen Transaktions-Namens als Überweisungszweck (**contract_purpose=`echo "${trx_file}"|sha256sum|cut -d ' ' -f1**)
- der Typ der Transaktion ist „partial“.

Abschließend wird noch ein alias namens „**send_trx**“ definiert für den Aufruf von ucs_client.sh. Dies ist letztendlich die Aktion, welche ausgelöst wird.

SCHRITT 3: STARTEN SIE DEN CONTRACTOR

Entweder führen Sie den **ucs_contractor.sh** manuell aus oder richten hierfür einen Job ein, z.B. mit CRON. Um Ihren Smart Contract auszuführen, übergeben Sie einfach Ihr Ruleset und Ihren Smart Contract und mit vollständigem Pfad:

```
./ucs_contractor.sh -ruleset /pfad/zu/contract.ruleset
-contract /pfad/zu/contract.logic
```

Bitte beachten Sie, dass bei **accountant.logic** keine Ausgabe erfolgt, wenn keine Transaktion dem Rulesets entspricht.
Um den Hilfetext anzuzeigen, führen Sie Folgendes aus:

```
./ucs_contractor.sh -help
```

Weitere informationen:

Bei der Ausführung von **ucs_contractor.sh** checkt das Script ob eine Contract-Datei vorhanden ist (Parameter **-contract <PFAD>**) und ob die Ruleset-Datei vorhanden ist (Parameter **-ruleset <PFAD>**). Wenn beide Dateien vorhanden sind wird die Funktion **contract_action()** der Contract-Datei gesourced und aufgerufen. Das war's schon.

Wenn Sie eine Ruleset-Datei benötigen muss die Logik zum sourcen/lesen dieser Datei innerhalb der Contract-Datei vorhanden sein (siehe **accountant.logic** und **tombola.logic**). Die Ruleset-Datei wird **NICHT** innerhalb des Scriptes **ucs_contractor.sh** gesourced. Der Contractor läd lediglich die Logik der Contract-Datei und ruft die Funktion darin auf. Diese Logik können Sie frei gestalten. Sämtliche Trigger und Aktionen müssen in einer Funktion namens **contract_action()** definiert werden.

Unter Umständen benötigen Sie für das was Sie realisieren wollen keine Ruleset-Datei, aber **ucs_contractor.sh** überprüft trotzdem ob eine entsprechende Datei vorhanden ist. Eine Lösung wäre denselben Pfad bei **-ruleset** als bei **-contract** zu übergeben.

Die Tatsache, dass die Contract-Logik und das Ruleset verschiedene Dateien sind erlaubt es dem Benutzer denselben Contract mit verschiedenen Rulesets aufzurufen!

10. UCA-LINK Server

UCS bietet Benutzern die Möglichkeit eigene UCA Link Server zu betreiben. Diese Server agieren als Netzwerkknoten, die zur Verbreitung von Transaktionswissen beitragen indem sie den Synchronisierungsprozess zwischen den Benutzern automatisieren.

UCA-LINK SERVER INSTALLATION

SCHRITT 1 : DOWNLOADEN SIE DEN QUELLCODE

Zuerst müssen Sie sich D. J. Bernstein's **ucspi-tcpserver** besorgen (siehe http://cr.yp.to/ucspi-tcp.html). Es gibt mehrere Möglichkeiten, es zum Laufen zu bringen. Während Sie mit **`make`** einen eigenen Build erstellen können, haben wir einfach das Debian-Paket **ucspi-tcp-ipv6** über **`apt-get`** installiert.

Nachdem Sie den ucs_client installiert haben gehen Sie in dieses Verzeichnis und entpacken Sie die Server Dateien:

```
tar -xvf server.tar
```

Die folgenden Dateien werden extrahiert:

```
control/server.conf
controller.sh
logwatch.sh
filewatch.sh
start_server.sh
stop_server.sh
sender.sh
receiver.sh
```

Des Weiteren werden die Ordner **`/log/`** und **`/server/`** extrahiert. Im Ordner **`/log/`** werden die Logfiles des Servers abgelegt. Im Ordner **`/server/`** werden die Tempfiles des Servers abgelegt.

SCHRITT 2 : PASSEN SIE DEN SERVER AN

Passen Sie die Datei **`control/server.conf`** an. Sie müssen dort mindestens die IP-Adresse Ihres Servers eintragen sowie die Daten des Benutzers (Kontoname, PIN, Passwort), der vom Server verwendet werden soll.

SCHRITT 3 : VERÖFFENTLICHEN SIE IHRE SERVER-DETAILS

Fügen Sie der Datei **`uca.conf`** im Ordner **`/control/`** eine Zeile hinzu. Das Format sollte sein:

```
IP_ODER_URL,SENDER_PORT,EMPFÄNGER_PORT,BESCHREIBUNG,
```

wie z.B.:

```
127.0.0.1,15000,15001,UCA LINK SERVER,
```

Senden Sie diese Informationen an Ihre Freunde und die Personen, die Ihren

Server verwenden möchten. Sie können die Details auch im UCS Forum oder anderswo im Internet veröffentlichen.

SCHRITT 4 : STARTEN SIE DEN SERVER

Sie starten den Server, indem Sie das Script **`start_server.sh`** ausführen:

```
./start_server.sh
```

Das Script ruft **`controller.sh`** auf, welcher als Daemon fungiert und die Scripte **`sender.sh`**, **`receiver.sh`**, **`logwatch.sh`** und **`filewatch.sh`** im Hintergrund ausführt und überwacht. Personen können sich jetzt automatisch mit Ihnen synchronisieren, indem Sie die UCA-Link-Funktionalität verwenden.

SCHRITT 5 : STOPPEN SIE DEN SERVER

Sie stoppen den Server, indem Sie das Script **`stop_server.sh`** ausführen:

```
./stop_server.sh
```

11. Universal Credit Webwallet

Das Webwallet ist eine einfach aufgebaute Lösung, um den Zugriff auf das Wallet über eine Webseite bereitzustellen. Es wurde auf einem Setup mit NGINX und PHP-FPM (FastCGI) entwickelt und getestet.

Die Benutzer starten auf einer Landingpage namens **`index.html`.** Die Anmeldeinformationen des Benutzers werden per POST-Methode an das Skript **`wallet.php`** gesendet, welches serverseitig ausgeführt wird. Das Skript selbst ruft dann ein Shell-Skript namens **`webwallet.sh`** auf. Das Skript **`webwallet.sh`** fungiert als Verbindung zwischen dem Wallet und dem Webserver. Das Webwallet-Skript triggert die Aufrufe von **`ucs_client.sh`**, fängt dessen Ausgabe ab und erstellt basierend auf diesen Daten eine Webseite für den Benutzer. Das Skript gibt im Wesentlichen HTML-Code an STDOUT aus, der dann an den Webserver weitergeleitet wird.

WEBWALLET INSTALLATION

SCHRITT 1 : INSTALLIEREN SIE NGINX UND PHP-FPM

Sobald Sie eine funktionierendes Setup haben, ist es wichtig, die in der NGINX-Konfiguration festgelegten *Timeouts* zu erhöhen. Fügen Sie Ihrer NGINX-Serverkonfiguration die folgenden Zeilen hinzu:

```
proxy_read_timeout 300;
proxy_connect_timeout 300;
proxy_send_timeout 300;
```

In der PHP-Sektion der NGINX-Serverkonfiguration müssen Sie die folgende Zeile direkt unter der Zeile hinzufügen, die **`fastcgi_pass`** enthält:

```
fastcgi_read_timeout 300s;
```

Diese Timeout-Werte können Sie festlegen wie sie wollen, aber denken Sie daran, dass es abhängig von Ihrer Hardware einige Minuten dauern kann bis das Skript alles berechnet hat. Um also ein Server-Timeout zu vermeiden, empfehlen wir, diesen Wert auf einige Minuten zu setzen. Sie müssen auch sicherstellen, dass der Benutzer, unter dem PHP ausgeführt wird, Schreibzugriff auf das Home-Verzeichnis des Wallet hat, da im Rahmen der Upload-Funktionalität Dateien in diesen Ordner hochgeladen werden.

SCHRITT 2 : EXTRAHIEREN SIE DEN QUELLCODE

Gehen Sie in das Wallet-Home-Verzeichnis und entpacken Sie die Datei **`webwallet_home.tar`:**

```
tar -xvf webwallet_home.tar
```

Danach extrahieren Sie die Datei **`webwallet_www-data.tar`**. Der Zielordner, der im folgenden Befehl verwendet wird, ist Ihr Webserver-Verzeichnis (**`/var/www/html`**). Wenn Ihr Webserver ein anderes Verzeichnis verwendet,

müssen Sie den Pfad nach der Option „-C“ auf den Pfad ändern, den Ihr Webserver verwendet (**stellen Sie sicher, dass Sie Schreibrechte für dieses Verzeichnis haben!** Wenn Sie diese Berechtigungen nicht haben, verwenden Sie **`sudo`**):

```
tar -xvf webwallet_www-data.tar -C /var/www/html
```

SCHRITT 3 : FÜHREN SIE DAS INSTALLSKRIPT AUS

Führen Sie nun das Installationsskript aus (der Benutzer, der dieses Skript ausführt, muss Schreibzugriff auf das Inhaltsverzeichnis des Webservers haben, z verwenden, wenn es anders ist):

```
sudo ./install_webwallet.sh /var/www/html
```

SCHRITT 4 : STARTEN SIE NGINX UND PHP-FPM

Starten Sie NGINX und PHP-FPM und Sie sollten über den Browser auf das Webwallet zugreifen können.