Universal Credit System

documentación técnica

¡Bienvenido!

En este documento encontrará documentación técnica detallada **del Universal Credit System**. Contiene toda la información que necesitas para poder implementar y/o adaptar el programa a tus necesidades.

Contenido

1. Dependencias

El programa depende de otros programas que deben instalarse. Durante la instalación, el script **`install.sh`** realizará una comprobación si falta algún programa. Se utilizan los siguientes programas:

nombre	**usado para**
awk	ordenar/filtrar datos
basename	eliminar directorio y sufijo de nombres de archivos
bc	cálculos de punto flotante
cat	concatenar contenido
chmod	cambiar permisos de archivo/directorio
cp	copiar archivos
curl	enviar consulta a TSA y solicitar respuesta
cut	extraer datos de flujos de datos
date	operaciones de fecha
dd	convertir archivos
dialog	interfaz gráfica de usuario
dirname	eliminar el sufijo que no es de directorio del nombre del archivo
echo	escribir salida
find	buscar archivos/directorios
flock	administrar bloqueos de lectura para configuraciones multiusuario
gpg	firma de transacciones
grep	Escanear archivos/transmisiones en busca de patrones
head	mostrar líneas de encabezado de un archivo
ls	listar archivos y directorios
mkdir	crear carpetas y subcarpetas
mv	mover archivos
netcat	enviar/solicitar archivos
openssl	Verificación del sello de la TSA
printf	escribier salida
rm	eliminar archivos
sed	leer/modificar texto
sha224sum	archivos hash
sha256sum	archivos hash
sort	ordenar archivos
stat	obtener permisos de archivo/directorio
tail	mostrar las líneas finales de un archivo
tar	crear un archivo de transacción o un archivo de sincronización
test	archivos de prueba
touch	crear archivos
tr	convertir caracteres
umask	determinar umask
uniq	filtrar archivos
wc	contar líneas, palabras, bytes
wget	recuperar archivos de certificados de TSA desde Internet

2. Carpetas

nombre	descripción
~/assets/	**contiene los assets creados por los usuarios**
~/backup/	**contiene las copias de seguridad creadas por el usuario**
~/certs/	**contiene subcarpetas para TSA con certificados**
~/control/	**contiene archivos importantes del sistema**
~/keys/	**contiene los archivos de clave pública de los usuarios.**
~/lang/	**contiene los archivos de idioma**
~/proofs/	**contiene subcarpetas para las proofs de los usuarios**
~/theme/	**contiene los temas para GUI**
~/trx/	**contiene las transacciones de todos los usuarios.**
~/userdata/	**contiene archivos temporales relacionados con el usuario**

3. Detalles

~/assets/

Contiene los assets que han sido creados por los usuarios. Los assets pueden considerarse tokens. Los archivos de activos se denominan así:

<ASSET_SIMBOLO>.<MARCADETIEMPO>

<ASSET_SIMBOLO> es una cadena. *<MARCA_TIEMPO>* es el sello en formato de marca de tiempo UNIX (hora de época de Unix).

~/backup/

Contiene copias de seguridad del sistema activadas por el usuario. Con estos archivos de respaldo, un usuario puede restaurar sus datos si están dañados. Los archivos de respaldo se denominan así:

<MARCADETIEMPO>.bc

<MARCA_TIEMPO> es el sello en formato de marca de tiempo UNIX (hora de época de Unix).

~/certs/

Esta carpeta contiene una subcarpeta para cada TSA. En esta subcarpeta se encuentran los certificados iniciales de las TSA. Actualmente sólo se admite

freeTSA. Los archivos son:

cacert.pem	(Certificado de CA)
tsa.pem	(Certificado de TSA)

Estos certificados se utilizan para verificar los archivos TSA que se encuentran en el directorio ~**/proofs/** a través de **`openssl ts`**.

~/control/

Esta carpeta contiene archivos importantes del sistema que utiliza UCS.

Archivos en esta carpeta:

config.conf	archivo de configuración
uca.conf	UCA lista
tsa.conf	TSA lista
install_config.conf	archivo de configuración predeterminado
dh.db	base de datos para Perfect Forward Secrecy
install.dep	dependencias de instalación
keyring.file	GPG keyring
HELP.txt	texto de ayuda

Todas las claves privadas de las cuentas que se han creado en esta máquina se guardan en la carpeta **~/control/keys/**.

~/keys/

Esta carpeta contiene las claves públicas de todos los usuarios. Las claves son claves RSA de 4096 bits. La creación de estas claves se realiza a través de GnuPG, mientras que todas las claves se administran en ***keyring.file*** en el directorio ***~/control/*** de scripts. Las claves se nombran de la siguiente manera:

<CADENA_DE_TEXTO>.<MARCADETIEMPO>

<CADENA_DE_TEXTO> es en realidad un hash SHA-256 que se calcula aplicando el hash a la cadena "*<NOMBREDECUENTA>*_*<CODIGOPIN>*_*<MARCADETIEMPO>*", mientras que <MARCADETIEMPO> es el punto de creación en formato de marca de tiempo UNIX (hora de época de Unix). El cálculo del hash también se utiliza durante el inicio de sesión. El nombre de la cuenta y el PIN se utilizan junto con los sellos para encontrar una clave coincidente.

~/lang/

Los archivos de nuevos idiomas se importarán automáticamente y se podrán seleccionar en la GUI. Se obtendrán dentro del guión al principio.

La convención de nomenclatura de los archivos lang es:

lang_<idioma-corto>_<idioma-largo>.conf

<language-corte> es el código del país, p.e. ***EN*** para ***ENGLISH*** o ***FR*** para ***FRANCE***, mientras que *<language-largo>* es el nombre del idioma en el idioma extranjero, p. ***SVENSKA*** para *SUECO* o ***ITALIANO*** para *ITALIANO*. Puede agregar nuevos archivos a esta carpeta y el script los incluirá automáticamente en la lista de idiomas disponibles.

¡Asegúrate de no cambiar/eliminar las etiquetas '<tag>' o '/n'! ¡Estas etiquetas se utilizan para formatear la interfaz gráfica de usuario proporcionada por el `dialog` !

Todo lo demás se puede adoptar al idioma relacionado. Si los archivos de nuevos idiomas tienen nombres como se describe anteriormente, el script puede reconocerlos y usted puede seleccionarlos en la GUI.

~/proofs/

Esta carpeta contiene una subcarpeta para cada usuario y en esta subcarpeta están las proofs de los usuarios (archivo índice, consulta TSA, respuesta TSA).

Las subcarpetas están en el siguiente formato (por usuario):

/<DIRECCIÓN>/

<DIRECCIÓN> es igual al usuario. La verificación de los archivos TSA de los usuarios en esta carpeta se realiza mediante el comando **`openssl ts`**:

freetsa.tsq	consulta gratuita de freeTSA
freetsa.tsr	respuesta gratuita de freeTSA
<DIRECCIÓN>.txt	archivo de índice

El archivo de índice contiene una lista de todos los usuarios reconocidos con sus pruebas y transacciones. El archivo de índice si se verifica usando GnuPG.

~/theme/

Esta carpeta contiene los temas utilizados por el comando de `dialog`. Cualquier tema colocado en esta carpeta será reconocido automáticamente y estará disponible en el **menú principal → configuración → temas**.

~/trx/

Esta carpeta contiene las transacciones de todos los usuarios. La convención de nomenclatura de transacciones:

<DIRECCIÓN>.<MARCADETIEMPO>

<DIRECCIÓN> es la dirección del remitente mientras que *<MARCADETIEMPO>* es la fecha de creación de la transacción en formato de marca de tiempo UNIX (hora de época de Unix). Las transacciones son archivos de texto simples que contienen un encabezado que incluye toda la información relacionada y una firma del usuario que las envía.

~/userdata/

Contiene una subcarpeta para cada usuario que inició sesión en esta máquina. En esta subcarpeta archivos temporales específicos del usuario:

all_assets.dat	lista de todos los assets
all_keys.dat	lista de todas las claves
all_accounts.dat	lista de todas las cuentas
all_trx.dat	lista de todas las transacciones
blacklisted_accounts.dat	lista de todas las cuentas eliminadas
blacklisted_trx.dat	lista de todas las transacciones eliminadas
depend_accounts.dat	lista de todas las cuentas dependientes
depend_trx.dat	lista de todas las transacciones dependientes
depend_confirmations.dat	lista de todas las transacciones dependientes sin suficientes confirmaciones
<YYYYMMDD>_ledger.dat	archivo de libro mayor de la fecha correspondiente
<YYYYMMDD>_scoretable.dat	archivo de puntuación de la fecha correspondiente
<YYYYMMDD>_index_trx.dat	lista de todas las transacciones que el usuario ha reconocido en la fecha correspondiente

Los archivos que se han extraído de archivos de transacciones o archivos de sincronización se almacenan primero en ***~/userdata/<NOMBRE_USUARIO>/temp/***. Desde allí, se mueven a las carpetas relacionadas o se eliminan.

4. Assets

El Universal Credit System apoya la creación y el uso de *tokens*, aquí llamados **assets**. Los assets pueden ser fungible o non-fungible. Todos los assets se almacenan en **~/assets/**. La principal diferencia entre la moneda UCC y los activos es que **no se realiza ninguna puntuación** al transferir assets.

FUNGIBLE ASSETS

Fungible assets son assets que todos los usuarios pueden convertir. Esto significa que cada usuario puede canjear su saldo por este asset. El intercambio de *fungible assets* es ilimitado; no hay una cantidad máxima que se puede cambiar/convertir. A continuación se muestra un ejemplo de un fungible asset ficticio `'TestFungibleToken'` que tiene el símbolo de activo `'TFT.1655676000'`:

asset_description=TestFungibleToken
asset_price=2.000000000
asset_fungible=1

asset_description es la descripción de los asset, *asset_price* es el precio por unidad en UCC. *asset_fungible=1* define que se trata de un activo fungible. La importación de fungible assets que otros usuarios han creado **está deshabilitada de forma predeterminada**. Para habilitar la importación automática de activos fungibles, debe modificar el archivo **~/control/config.conf** y configurar **'import_fungible_assets=1'**.

NON-FUNGIBLE ASSETS

Non-fungible assets son assets que no se pueden convertir ni intercambiar. Para poseer un activo de este tipo, debe recibirlo de un propietario o debe ser usted mismo el propietario inicial. El importe total de un non-fungible assets se define como valor **asset_quantity**. A continuación se muestra un ejemplo de un activo ficticio no fungible `'TestNonFungibleToken'` que tiene el símbolo de activo `'TNFT.1655676000'`:

asset_description=TestNonFungibleToken
asset_owner=9d8c98a97b2c3e689afef90310a35130bde86fd6f43ef6764b391c40ba37f8dd.1613477808
asset_quantity=100.000000000
asset_fungible=0

asset_description es la descripción de los assets, *asset_owner* es el propietario inicial, *asset_quantity* es la cantidad de tokens. *asset_fungible=0* define que se trata de un non-fungible asset. La importación de non-fungible assets que otros usuarios hayan creado **está deshabilitada de forma predeterminada**. Para habilitar la importación automática de activos no fungibles, debe modificar el archivo **~/control/config.conf** y configurar **'import_non_fungible_assets=1'**.

5. Transacción de ejemplo

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA512

:TIME:1719848263

```
:AMNT:1.000000000
:ASST:UCC
:SNDR:ca2c6f1d030c0ea7e56893a89c32d6c86478b56ff40cfb327608ef47a58bc401.1613477644
:RCVR:9d8c98a97b2c3e689afef90310a35130bde86fd6f43ef6764b391c40ba37f8dd.1613477808
:PRPS:
hQIMA0/utTbFeaUFAQ/+MPW78a86dOAaeH1u5zuwIzfEoW1KHVrDvALeJUQ3DPAx
5rZoXEKdJ6sVtxb2NKJHmmsZoYYTPcrIY68P/Ur27mYz4uUz/T5C5Rt9vsvZ4Umg
DaHR5VIFdXZ+uDYZNrRLUGSsYPaSQOqHfTh8tG4+6RU3ygYu4s/kNCU6F/soVPVM
[...]
-----BEGIN PGP SIGNATURE-----

iQIzBAEBCgAdFiEEWbstlJvHpx/H2ElhXtJKJq/t0uYFAmaCzUcACgkQXtJKJq/t
0uZc2w//SEWaklJbwgaisIT0cUpbskzRPX7V5aTPMYPlmbyFwmM8/sMqq3XrO+Zd
nJ/uzbvAu5e74QBmUpoN4kJgrJeTf+kH+X6YLMq9rVhxfPxGSfwr9g2P5hcssWZ9
[...]
-----END PGP SIGNATURE-----
```

Una transacción es en realidad un archivo de texto claramente firmado en formato OpenPGP. Contiene toda la información necesaria: *la fecha de creación* (TIME), *el monto a transferir* (AMNT), *la definición del asset* (ASST), *el remitente de la transacción* (SNDR), *el destinatario de la transacción* (RCVR) y *un propósito cifrado* (PRPS).

6. Instalación de wallet

Suponiendo que está utilizando la herramienta de empaquetado APT, se utiliza el comando **`apt-get install`**. Tenga en cuenta que si está utilizando cualquier herramienta de empaquetado que no sea APT, el comando para instalar un paquete puede ser diferente. En este caso, cambie **`apt-get install`** por el

comando que está utilizando su herramienta de empaquetado.

Instale Git (tal vez necesite usar **sudo** al frente):

```
apt-get install git
```

Crea un directorio donde quieras y accede a este directorio:

```
mkdir ucs
cd ucs
```

Clona el repositorio de GitHub y accede a este directorio:

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

Ahora puede ejecutar el script **install.sh**. El script buscará programas dependientes y, si todos los programas dependientes están instalados, la instalación continuará. Si es necesario instalar un programa, el script generará los nombres del programa y luego saldrá. En este caso, primero debe instalar estos programas y luego ejecutar el script install.sh nuevamente:

```
./install.sh
```

Después de la instalación, puede ejecutar **ucs_client.sh**:

```
./ucs_client.sh
```

7. Modo de línea de comandos

A continuación puede encontrar ejemplos detallados de comandos para el modo cmd. Con estos comandos es muy fácil configurar soluciones de pago automatizadas.

COMO CREAR UN USARIO

```
COMANDO DE EJEMPLO:
./ucs_client.sh -action create_user -user TESTUSER -password
TESTPASSWORD

SALIDA DE PANTALLA:
USER:<NOMBREDECUENTA>
PIN:<CODIGOPIN>
PASSWORD:>CONTRASENA<          # NOTE: CONTRASENA PUT IN ><
ADRESS:<DIRECCIÓN>
KEY:<ARCHIVO_CLAVE>
KEY_PUB:/keys/DIRECCIÓN
KEY_PRV:/control/keys/DIRECCIÓN
```

NOTA: Actualmente, la clave privada exportada siempre se almacena en la carpeta ~/control/keys, mientras que la clave pública siempre se almacena en la carpeta ~/keys. Entonces, si entrega una ruta donde se deben almacenar estas claves, ¡no se usará! Estas claves exportadas son sus claves de respaldo públicas y privadas; ¡será mejor que las guarde debajo de la almohada! Con estas claves y tus pruebas podrás restaurar tu cuenta si todo se pierde.

CÓMO CREAR UNA PEQUEÑA TRANSACCIÓN (solo empaquete archivos nuevos, si es posible)

```
COMANDO DE EJEMPLO:
./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
```

NOTA: Escriba "partial" significa que el programa verificará si el remitente y el destinatario tienen conocimiento de transacciones compartido y, de ser así, solo agregará datos al archivo de transacciones que sean nuevos para el remitente. Esto puede reducir el tamaño de un archivo de transacción.

CÓMO CREAR UNA TRANSACCIÓN GRANDE (empaquetar todos los archivos)

```
COMANDO DE EJEMPLO:
./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
```

NOTA: Escriba "completo" significa que empaquetará todos los datos independientemente de cualquier conocimiento de transacción compartido.

CÓMO LEER PARCIALMENTE UN ARCHIVO DE TRANSACCIÓN (solo descomprimir archivos nuevos)

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

NOTA: Escriba "partial" significa que el programa verificará si el remitente y el destinatario tienen conocimiento de transacciones compartido y, de ser así, solo descomprimirá los datos que sean nuevos para el remitente. Esto es estándar y siempre debes hacerlo de esta manera para evitar que se sobrescriban otros archivos que ya tienes.

CÓMO LEER COMPLETAMENTE UN ARCHIVO DE TRANSACCIÓN (desempaquetar todos los archivos y sobrescribir los existentes):

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

NOTA: Type "full" significa que el programa descomprime todos los datos del archivo de transacciones. ¡Esto anula sus datos existentes y solo debe hacerse con mucha precaución y conciencia! Por ejemplo, esto le permite restaurar sus datos mediante un archivo de transacción sólo si está dañado. ¡CUIDADO CON ESTO!

CÓMO CREAR UN ARCHIVO DE SINCRONIZACIÓN (contiene todos los archivos):

```
COMANDO DE EJEMPLO:
./ucs_client.sh -action create_sync -user TESTUSER -pin 12345 -
password TESTPASSWORD -path /path/to/outputdir
```

NOTA: Como no existe un receptor explícito para un archivo de sincronización, siempre contiene todos los datos de todos los usuarios. Depende del receptor del fichero qué datos extraer (full o partial).

CÓMO LEER PARCIALMENTE UN ARCHIVO DE SINCRONIZACIÓN (solo descomprimir archivos nuevos):

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

NOTA: Type "partial" significa que el programa verificará si el remitente y el destinatario tienen conocimiento de transacciones compartido y, de ser así, solo descomprimirá los datos que sean nuevos para el remitente. Esto es estándar y siempre debes hacerlo de esta manera para evitar que se sobrescriban otros archivos que ya tienes.

CÓMO LEER COMPLETAMENTE UN ARCHIVO DE SINCRONIZACIÓN (desempaquetar todos los archivos y sobrescribir los existentes):

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

NOTA: Type "full" significa que el programa descomprime todos los datos del archivo de sincronización. ¡Esto anula sus datos existentes y solo debe hacerse con mucha precaución y conciencia! Por ejemplo, esto le permite restaurar sus datos mediante un archivo de sincronización sólo si está dañado. ¡CUIDADO CON ESTO!

CÓMO SINCRONIZAR CON LA UCA

```
COMANDO DE EJEMPLO:
./ucs_client.sh -action sync_uca -user TESTUSER -pin 12345 -
password TESTPASSWORD
```

NOTA: La acción "sync_uca" no creará ninguna salida si tiene éxito y siempre saldrá con el código 0 incluso si falló la recepción/envío de datos a las UCA definidas. Si falla la recepción/envío a una UCA más, generará un mensaje de “ERROR” que contiene la IP utilizada (<uca_ip>) y el puerto (<ucs_snd_port>) como se define en ~/control/uca.conf.

CÓMO CREAR UNA COPIA DE SEGURIDAD:

```
COMANDO DE EJEMPLO:
./ucs_client.sh -action create_backup
```

CÓMO RESTAURAR UNA COPIA DE SEGURIDAD:

```
COMANDO DE EJEMPLO:
./ucs_client.sh -action restore_backup -path
/path/to/ucs/backup/<STAMP>.bcp
```

CÓMO MOSTRAR ESTADÍSTICAS:

```
COMANDO DE EJEMPLO:
./ucs_client.sh -action show_stats -user TESTUSER -pin 12345 -
password TESTPASSWORD
```

8. Universal Credit Contractor

El Contractor actúa como un script contenedor para el script **`ucs_client.sh`** y permite al usuario configurar contratos inteligentes para transacciones. Cuando se ejecuta, el script bash **`ucs_contractor.sh`** leerá la lógica de un archivo y realizará acciones basadas en la lógica y el conjunto de reglas.

¿QUÉ ES UN CONTRACT?

Un contrato siempre consta de **al menos** dos archivos:

- un archivo que contiene la lógica real en la carpeta **`/contracts/`** (que contiene una definición de una función llamada **`contract_action()`** que se carga mediante el script)

- un archivo que contiene el conjunto de reglas en la carpeta **`/rulesets/`** (definiciones de variables utilizadas por la lógica)

Ambos archivos se entregan a **`ucs_contractor.sh`** mediante parámetros. Durante la ejecución, la lógica del contrato se carga y controla con las variables del archivo de conjuntos de reglas. En principio, se puede implementar cualquier lógica.

CÓMO INSTALAR Y CONFIGURAR

PASO 1 : DESEMPAQUE EL CÓDIGO FUENTE

Para ejecutar el contratista, necesita tener un cliente completo configurado con un usuario. Suponiendo que ya tiene el ucs_client, acceda al directorio del cliente y descomprima al contratista:

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

El tarball contiene los siguientes archivos/carpetas:

- script **`ucs_contractor.sh`**
- carpeta **`/contracts/`**
- carpeta **`/rulesets/`**
- archivo **`/control/contractor_HELP.txt`**

El tarball también contiene algunos contracts de ejemplo (cashier, filter, accountant y tombola) y conjuntos de rulesets relacionados para estos contracts.

PASO 2 : DEFINA SU(S) CONTRACT(S)

Cree una lógica de smart contract y un conjunto de reglas basadas en sus necesidades.

EJEMPLO

El siguiente ejemplo es un conjunto de reglas para el smart contract **`accountant.logic`** proporcionado. El smart contract **`accountant.logic`** actúa como un simple contador que envía transacciones en función de las transacciones recibidas. Sólo los parámetros relacionados con la transacción se pueden definir como desencadenantes y la acción se limita a la creación de

nuevas transacciones.

Vea a continuación el ejemplo de conjunto de ruleset **`accountant.ruleset`**:

```
ruleset_asset="TU_ASSET_AQUÍ"
ruleset_sender="*"
ruleset_receiver="TU_DIRECCIÓN_AQUÍ"
ruleset_amount="*"
ruleset_amount_comparison_operator=""
ruleset_amount_comparison_variable=""
ruleset_purpose="*"
ruleset_required_confirmations=0
ruleset_payment_data="RUTA_TO_ARCHIVO_DE_HISTORIAL_DE_PAGOS"

contract_asset="${trx_asset}"
contract_sender="TU_DIRECCIÓN_AQUÍ"
contract_sender_password="TU_CONTRASEÑA_AQUÍ"
contract_receiver="${trx_sender}"
contract_amount="${trx_amount}"
contract_purpose="${trx_file}"
contract_type="partial"
```

El conjunto de ruleset anterior garantiza que el smart contract accountant.logic envíe todas las transacciones que le fueron enviadas al remitente (si ingresa un asset, su dirección y su contraseña).

`accountant.logic` buscará transacciones:

- coincidente con el asset definido (**`ruleset_asset="TU_ASSET_AQUÍ"`**)
- no importa quién lo envió (**`ruleset_sender="*"`**)
- usted como receptor (**`ruleset_receiver="TU_DIRECCIÓN_AQUÍ"`**)
- cualquier cantidad (**`ruleset_amount="*"`**)
- cualquier propósito (**`ruleset_purpose="*"`**)
- que no tienen confirmaciones (**`ruleset_required_confirmations=0`**)
- **`ruleset_payment_data`** es la ruta completa a un archivo vacío que se utiliza para guardar los nombres de archivos de transacciones ya procesadas (historial)

Si una o varias transacciones coinciden con este criterio, el contratista creará una transacción:

- tener el asset inicial que fue enviado como asset para enviar (**`contract_asset="${trx_asset}"`**)
- tener el monto inicial que se envió como monto a enviar (**`contract_amount="${trx_amount}"`**)
- el remitente inicial de la transacción como receptor (**`contract_receiver="${trx_sender}"`**)
- tener el nombre de archivo de la transacción inicial como propósito (**`contract_purpose="${trx_file}”`**
- type de transacción es "partial"

PASO 3 : PROGRAMAR EL CONTRACTOR

Puede ejecutar manualmente **ucs_contractor.sh** o programar un trabajo para esto con CRON, por ejemplo. Para ejecutar su contrato simplemente entregue su conjunto de ruleset y su contract con la ruta completa:

```
./ucs_contractor.sh -ruleset /path/to/contract.ruleset -
contract /path/to/contract.logic
```

Tenga en cuenta que **accountant.logic** no generará resultados en pantalla si ninguna transacción coincide con el conjunto de reglas.

Para mostrar el texto de ayuda ejecute:

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

Mas informatión:
Durante la ejecución, el script **ucs_contractor.sh** comprobará si hay un archivo de contrato (parámetro **-contract <PATH>**) y si hay un archivo de conjunto de reglas (parámetro **-ruleset <PATH>**). Si ambos archivos están ahí, se obtendrá y llamará a la función **contract_action()** del archivo lógico del contrato. Eso es todo.

Esto significa que si necesita un archivo de conjunto de reglas, la lógica para obtenerlo/leerlo debe colocarse dentro de la lógica del contrato (consulte **accountant.logic** y **tombola.logic**). Por lo tanto, el archivo del conjunto de reglas **NO** se lee dentro de **ucs_contractor.sh**. El contratista solo carga el archivo lógico y llama a la función dentro de ese archivo. Este archivo lógico puede contener lo que quieras. Todos los desencadenantes y acciones deben definirse en una función denominada **contract_action()**.

Dependiendo de lo que desee hacer, es posible que no necesite un archivo de ruleset, pero ucs_contractor.sh aún así verificará si ese archivo está allí. Una solución sería entregar la misma ruta para **-ruleset** que para **-contract**.

El hecho de que la lógica del contract y el ruleset sean archivos separados permite al usuario ejecutar la misma lógica del contract con diferentes ruleset.

9. Universal Credit Authority Link Server

UCS permite al usuario configurar sus propios servidores UCA Link. Los servidores UCA Link pueden considerarse nodos que ayudan a difundir el conocimiento de las transacciones al automatizar el proceso de sincronización.

PASO 1 : OBTÉN EL CÓDIGO FUENTE

Primero que nada tienes que conseguir el **ucspi-tcpserver** (ver http://cr.yp.to/ucspi-tcp.html) de D. J. Bernstein. Hay varias formas de hacerlo funcionar. Si bien puede crear su propia compilación usando el comando **make**, hemos instalado el paquete Debian **ucspi-tcp-ipv6** mediante **apt-get**. Después de haber instalado el cliente, vaya a este directorio y extraiga los archivos del servidor:

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

Se extraerán los siguientes archivos:

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

También se extraerán las carpetas **~/log/** y **~/server/**. La carpeta **~/log/** es donde el servidor escribirá los archivos de log. En la carpeta **~/server/** se almacenan los archivos temporales del servidor.

PASO 2 : CONFIGURA EL SERVIDOR

Modifique el archivo **~/control/server.conf**. Al menos debe ingresar su IP y las credenciales de inicio de sesión (nombre de usuario, código PIN, contraseña) del usuario que debe utilizar el servidor.

PASO 3 : PUBLICAR LOS DETALLES DE SU SERVIDOR UCA LINK

Agregue una línea al archivo **~/control/uca.conf**

El formato debe ser:

```
IP_O_URL,ENVIAR_PORTAR,RECIBIR_PUERTO,DESCRIPCIÓN,
```

como por ejemplo:

```
127.0.0.1,15000,15001,SERVIDOR PERSONALIZADO,
```

Envía esto a tus amigos y a las personas que quieras que utilicen tu servidor. También puede publicar los detalles en este foro o en cualquier otro lugar de la web.

PASO 4 : INICIAR EL SERVIDOR

Inicia el servidor ejecutando el script **`start_server.sh`**:

`./start_server.sh`

El script iniciará **`controller.sh`** que actúa como demonio ejecutando y monitoreando los scripts **`sender.sh`**, **`receiver.sh`**, **`logwatch.sh`** y **`filewatch.sh`** en segundo plano. Las personas ahora pueden sincronizarse automáticamente con usted mediante la función de UCA Link.

PASO 5 : DETENER EL SERVIDOR

Detiene el servidor ejecutando el script **`stop_server.sh:`**

`./stop_server.sh`

10. Universal Credit Webwallet

Webwallet es una solución sencilla para proporcionar acceso a la billetera a través de una página web. Fue desarrollado y probado en una configuración con NGINX y PHP-FPM.

Los usuarios comienzan en una página de destino denominada **`index.html`**.

Las credenciales de los usuarios se envían mediante el método POST al script **wallet.php** que se ejecuta en el lado del servidor. Luego, el script en sí llama a un script de shell llamado **webwallet.sh.** El script **webwallet.sh** actúa como conector entre el cliente de billetera y el servidor web. El script del webwallet activa las llamadas de **ucs_client.sh**, captura el resultado y crea una página web para el usuario basada en esos datos. Básicamente, el script genera código html en *STDOUT* que luego se reenvía al servidor web.

WEBWALLET INSTALLATION

PASO 1 : INSTALA Y CONFIGURA NGINX CON PHP-FPM

Una vez que tenga una configuración funcional, es importante aumentar los *timouts* que están configurados en la configuración de NGINX. Agregue las siguientes líneas a la configuración de su servidor NGINX:

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

En la sección PHP de la configuración del servidor NGINX, debe agregar la siguiente línea justo debajo de la línea que contiene **fastcgi_pass**:

```
fastcgi_read_timeout 300s;
```

Puede utilizar diferentes valores de *timeouts*, pero tenga en cuenta que, dependiendo de su hardware, el script podría tardar algunos minutos en calcular todo. Entonces, para evitar que se agote el tiempo de espera del servidor, sugerimos establecer este valor en unos minutos. También debe asegurarse de que el usuario bajo el cual se ejecuta PHP tenga acceso de escritura al directorio de inicio de la billetera porque los archivos se cargan en esta carpeta.

PASO 2 : DESEMPAQUE EL CÓDIGO FUENTE

Ingrese al directorio de inicio de wallet y descomprima **webwallet_home.tar**:

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

Después de eso, extraiga el archivo **webwallet_www-data.tar**. La carpeta de destino que se utiliza en el siguiente comando es el directorio de su servidor web (**/var/www/html**). Si su servidor web usa un directorio diferente, debe cambiar la ruta después de la opción '-C' a la que está usando su servidor web.

¡También debes asegurarte de tener permisos de escritura para este directorio! Si no tiene estos permisos, use **sudo** delante de este comando:

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

PASO 3 : EJECUTAR EL SCRIPT DE INSTALACIÓN

Ahora ejecute el script de instalación. El usuario que ejecuta este script debe tener acceso de escritura al directorio del servidor web, por ejemplo /var/www/html. Si no tiene permisos de escritura debe usar **sudo** al frente. Y tampoco olvide cambiar /var/www/html al directorio que está usando su servidor web si es diferente:

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

PASO 4 : INICIAR NGINX Y PHP-FPM

Inicie NGINX y PHP-FPM y debería poder acceder al webwallet a través del navegador.