#!/usr/bin/env perl

use strict;

use warnings;

use POSIX;

use Getopt::Std;

sub usage() {

 die("usage: $0 [-p api-port] dev-private-key [short-commit-hash]\n");

}

my %opt;

getopts('p:', \%opt);

usage() if @ARGV < 1 || @ARGV > 2;

my $port = $opt{p} || 12391;

my $privkey = shift @ARGV;

my $commit\_hash = shift @ARGV;

my $git\_dir = `git rev-parse --show-toplevel`;

die("Cannot determine git top level dir\n") unless $git\_dir;

chomp $git\_dir;

chdir($git\_dir) || die("Can't change directory to $git\_dir: $!\n");

open(POM, '<', 'pom.xml') || die ("Can't open 'pom.xml': $!\n");

my $project;

while (<POM>) {

 if (m/<artifactId>(\w+)<.artifactId>/o) {

 $project = $1;

 last;

 }

}

close(POM);

# Do we need to determine commit hash?

unless ($commit\_hash) {

 # determine git branch

 my $branch\_name = ` git symbolic-ref -q HEAD `;

 chomp $branch\_name;

 $branch\_name =~ s|^refs/heads/||; # ${branch\_name##refs/heads/}

 # short-form commit hash on base branch (non-auto-update)

 $commit\_hash ||= `git show --no-patch --format=%h`;

 die("Can't find commit hash\n") if ! defined $commit\_hash;

 chomp $commit\_hash;

 printf "Commit hash on '%s' branch: %s\n", $branch\_name, $commit\_hash;

} else {

 printf "Using given commit hash: %s\n", $commit\_hash;

}

# build timestamp / commit timestamp on base branch

my $timestamp = `git show --no-patch --format=%ct ${commit\_hash}`;

die("Can't determine commit timestamp\n") if ! defined $timestamp;

$timestamp \*= 1000; # Convert to milliseconds

# locate sha256 utility

my $SHA256 = `which sha256sum || which sha256`;

# SHA256 of actual update file

my $sha256 = `git show auto-update-${commit\_hash}:${project}.update | ${SHA256}`;

die("Can't calculate SHA256 of ${project}.update\n") unless $sha256 =~ m/(\S{64})/;

chomp $sha256;

# long-form commit hash of HEAD on auto-update branch

my $update\_hash = `git rev-parse refs/heads/auto-update-${commit\_hash}`;

die("Can't find commit hash for HEAD on auto-update-${commit\_hash} branch\n") if ! defined $update\_hash;

chomp $update\_hash;

printf "Build timestamp (ms): %d / 0x%016x\n", $timestamp, $timestamp;

printf "Auto-update commit hash: %s\n", $update\_hash;

printf "SHA256 of ${project}.update: %s\n", $sha256;

my $tx\_type = 10;

my $tx\_timestamp = time() \* 1000;

my $tx\_group\_id = 1;

my $service = 1;

printf "\nARBITRARY(%d) transaction with timestamp %d, txGroupID %d and service %d\n", $tx\_type, $tx\_timestamp, $tx\_group\_id, $service;

my $data\_hex = sprintf "%016x%s%s", $timestamp, $update\_hash, $sha256;

printf "\nARBITRARY transaction data payload: %s\n", $data\_hex;

my $n\_payments = 0;

my $is\_raw = 1; # RAW\_DATA

my $data\_length = length($data\_hex) / 2; # two hex chars per byte

my $fee = 0.001 \* 1e8;

die("Something's wrong: data length is not 60 bytes!\n") if $data\_length != 60;

my $pubkey = `curl --silent --url http://localhost:${port}/utils/publickey --data ${privkey}`;

die("Can't convert private key to public key:\n$pubkey\n") unless $pubkey =~ m/^\w{44}$/;

printf "\nPublic key: %s\n", $pubkey;

my $pubkey\_hex = `curl --silent --url http://localhost:${port}/utils/frombase58 --data ${pubkey}`;

die("Can't convert base58 public key to hex:\n$pubkey\_hex\n") unless $pubkey\_hex =~ m/^[A-Za-z0-9]{64}$/;

printf "Public key hex: %s\n", $pubkey\_hex;

my $address = `curl --silent --url http://localhost:${port}/addresses/convert/${pubkey}`;

die("Can't convert base58 public key to address:\n$address\n") unless $address =~ m/^\w{33,34}$/;

printf "Address: %s\n", $address;

my $reference = `curl --silent --url http://localhost:${port}/addresses/lastreference/${address}`;

die("Can't fetch last reference for $address:\n$reference\n") unless $reference =~ m/^\w{87,88}$/;

printf "Last reference: %s\n", $reference;

my $reference\_hex = `curl --silent --url http://localhost:${port}/utils/frombase58 --data ${reference}`;

die("Can't convert base58 reference to hex:\n$reference\_hex\n") unless $reference\_hex =~ m/^[A-Za-z0-9]{128}$/;

printf "Last reference hex: %s\n", $reference\_hex;

my $raw\_tx\_hex = sprintf("%08x%016x%08x%s%s%08x%08x%02x%08x%s%016x", $tx\_type, $tx\_timestamp, $tx\_group\_id, $reference\_hex, $pubkey\_hex, $n\_payments, $service, $is\_raw, $data\_length, $data\_hex, $fee);

printf "\nRaw transaction hex:\n%s\n", $raw\_tx\_hex;

my $raw\_tx = `curl --silent --url http://localhost:${port}/utils/tobase58/${raw\_tx\_hex}`;

die("Can't convert raw transaction hex to base58:\n$raw\_tx\n") unless $raw\_tx =~ m/^\w{255,265}$/; # Roughly 255 to 265 base58 chars

printf "\nRaw transaction (base58):\n%s\n", $raw\_tx;

my $sign\_data = qq|' { "privateKey": "${privkey}", "transactionBytes": "${raw\_tx}" } '|;

my $signed\_tx = `curl --silent -H "accept: text/plain" -H "Content-Type: application/json" --url http://localhost:${port}/transactions/sign --data ${sign\_data}`;

die("Can't sign raw transaction:\n$signed\_tx\n") unless $signed\_tx =~ m/^\w{345,355}$/; # +90ish longer than $raw\_tx

printf "\nSigned transaction:\n%s\n", $signed\_tx;

# Check we can actually fetch update

my $origin = `git remote get-url origin`;

die("Unable to get github url for 'origin'?\n") unless $origin && $origin =~ m/:(.\*)\.git$/;

my $repo = $1;

my $update\_url = "https://github.com/${repo}/raw/${update\_hash}/${project}.update";

my $fetch\_result = `curl --silent -o /dev/null --location --range 0-1 --head --write-out '%{http\_code}' --url ${update\_url}`;

die("\nUnable to fetch update from ${update\_url}\n") if $fetch\_result ne '200';

printf "\nUpdate fetchable from ${update\_url}\n";

# Flush STDOUT after every output

$| = 1;

print "\n";

for (my $delay = 5; $delay > 0; --$delay) {

 printf "\rSubmitting transaction in %d second%s... CTRL-C to abort ", $delay, ($delay != 1 ? 's' : '');

 sleep 1;

}

printf "\rSubmitting transaction NOW... \n";

my $result = `curl --silent --url http://localhost:${port}/transactions/process --data ${signed\_tx}`;

chomp $result;

die("Transaction wasn't accepted:\n$result\n") unless $result eq 'true';

my $decoded\_tx = `curl --silent -H "Content-Type: application/json" --url http://localhost:${port}/transactions/decode --data ${signed\_tx}`;

printf "\nTransaction accepted:\n$decoded\_tx\n";