Exercise 2: Verifying Digital Currency Transactions (simplified Bitcoin)
CIS4930-6930 Spring 2018

Assigned: 1/24/18
Due: 2/7/2018 (2 weeks)
Associated Readings: Ch.2 Overview of Transactions, Ch.6 Detailed

How a Bitcoin Transaction works:

What's a Transaction in Bitcoin?

The act of "spending" happens when a user digitally signs a transaction that transfers value from a previous transaction (an unspent transaction output, or UTXO) to a new owner. When a wallet has received bitcoin, the wallet has detected that a UTXO can be spent using the one of the keys controlled by that wallet. A user's bitcoin balance is the sum of all these UTXOs scattered on the blockchain. The wallet calculates the balance by scanning and aggregating the total value. Transaction outputs are indivisible chunks of bitcoin currency. An unspent output can only be consumed in its entirety by a transaction, so if you need change, then a transaction must produce a separate UTXO for change, so that all the UTXOs sum to the amount of the original value of the input. Most bitcoin transactions generate change. Once a UTXO has been used as input to another transaction, it has been spent and is no longer a UTXO.

Parts of a Bitcoin Transaction: Inputs, Outputs

Inputs – What you are spending – (Debit)
Transaction inputs reference previous transaction outputs and identify which UTXO is consumed and provide proof of ownership. To build a transaction, the wallet selects from the UTXOs it controls with enough value.

Outputs – the Result of your transaction – (Credit)
UTXOs "Unspent Transaction Outputs" are the fundamental building block of a transaction output. Transaction outputs are pairs that name the account (key) and the amount from the transaction inputs that are to be credited to the named account.

A transaction that does not provide proof of authorization to spend a UTXO listed as input, or that lists an invalid UTXO is not accepted. All accepted transactions are stored on the blockchain.

Undergrad/Grads: (Additional for Grads only further down)
Program Name: ledger

Program Description: Program to interact with a ledger. Your program shall provide an interactive menu. If in interactive mode, it will print suggestive prompts asking for input; in non-interactive mode, it will not print these prompts, it will just wait for a valid command. Input is terminated with a newline. The commands are either a single letter (case insensitive) or a complete word followed by a newline. If additional

input is required for a command, it shall be entered on a single line by itself.

The menu will contain:

[F]ile: Supply filename:<infilename>. Read in a file of transactions. Any invalid transaction shall be identified with an error message to stderr, but not stored. Print an error message to stderr if the input file named cannot be opened. The message shall be "Error: file <infilename> cannot be opened for reading" on a single line, where <infilename> is the name provided as additional command input.

[T]ransaction: Supply Transaction:<see format below> Read in a single transaction in the format shown below. It shall be checked for validity against the ledger, and added if it is valid. If it is not valid, then do not add it to the ledger and print a message to stderr with the transaction number followed by a colon, a space, and the reason it is invalid on a single line.

[E]xit: Quit the program

[P]rint: Print current ledger (all transactions in the order they were added) to stdout in the transaction format given below, one transaction per line.

[H]elp: Command Summary

[D]ump: Supply filename:<outfilename>. Dump ledger to the named file. Print an error message to stderr if the output file named cannot be opened. The message shall be "Error: file <outfilename> cannot be opened for writing" on a single line, where <outfilename> is the name provided as additional command input.

[W]ipe: Wipe the entire ledger to start fresh.

[I]nteractive: Toggle interactive mode. Start in non-interactive mode, where no command prompts are printed. Print command prompts and prompts for additional input in interactive mode, starting immediately (i.e., print a command prompt following the I command).

[V]erbose: Toggle verbose mode. Start in non-verbose mode. In verbose mode, print additional diagnostic information as you wish. At all times, output each transaction number as it is read in, followed by a colon, a space, and the result ("good" or "bad").

[B]alance: Supply username: (e.g. Alice). This command prints the current balance of a user.

Format of Transactions:
<TransID>; M; (<TransID>, <vout>)^M; N; (<AcctID>, <amount>)^N
Items in angle brackets are parameters, M and N are whole numbers, and caret M (or N) indicates M (or N) repetitions of the parenthesized pairs.

Example Transaction:
4787df35; 1;(f2cea539, 0);3; (Bob, 150)(Alice, 845)(Gopesh, 5)

<TransID> refers to a 32-bit transaction identifier given in hexadecimal format (4787df35 in the example). For now, it will just be given as input (later it will be calculated).

M is the number of UTXOs that are inputs to this transaction (1 in the example). The genesis transaction is the only transaction allowed to have zero input UTXOs, and must be the first transaction in any ledger.

<vout> refers to the value out (vout) index of the UTXO, where the first index is zero. For example, vout 0 in transaction f2cea539 refers to (Alice, 1000) (See ledger below).

Following the field with the number M of input UTXOs is the field that lists those UTXOs as a sequence of M pairs, each pair in parentheses, elements separated by a comma, consisting of a transaction ID and a vout from that transaction.

N is the number of transaction outputs (3 in the example). N is always positive.

<AcctID> refers to the alphanumeric account identifier (for now, we will just provide these).

<amount> is a natural number of satoshis that is credited to the account named in the pair by this transaction.

Following the field with the number N of value outputs is the field that lists those outputs as a sequence of N pairs, each pair in parentheses, elements separated by a comma, consisting of an account ID and an amount credited to that account by this transaction. The last such pair is the fee given to the account responsible for adding this transaction to the ledger. In this exercise, we will fill in the account identifier; later, this will be given as a keyword FEE in a proposed transaction, with the account owner responsible providing their account ID.

Example ledger:

f2cea539	0		1	(Alice, 1000)
4787df35	1	(f2cea539, 0)	3	(Bob, 150)(Alice, 845)(Gopesh, 5)
40671f57	1	(4787df35, 0)	3	(Gopesh, 100)(Bob, 45)(Bob, 5)
84dfb9b3	1	(40671f57, 0)	2	(Bob, 100)(Gopesh, 5)
f6847ad8	1	(40671f57, 1)	3	(Alice, 5)(Bob, 35)(Bob, 5)

File Format: One transaction per line with a carriage return to signify end of transaction. Semicolons delineate the table columns, commas separate values within the pairs, pairs are enclosed in parentheses.

File format version of example ledger:
```
f2cea539; 0; ; 1; (Alice, 1000)
4787df35; 1; (f2cea539, 0); 3; (Bob, 150)(Alice, 845)(Gopesh, 5)
40671f57; 1; (4787df35, 0); 3; (Gopesh, 100)(Bob, 45)(Bob, 5)
84dfb9b3; 1; (40671f57, 0); 2; (Bob, 100)(Gopesh, 5)
f6847ad8; 1; (40671f57, 1); 3; (Alice, 5)(Bob, 35)(Bob, 5)
```

This is both the format for input transactions and for the dumped ledger.

Processing:
We place no restriction on the internal storage mechanism you use.

Each transaction (line of input) must be processed to determine if it is well-formed (follows the format given). This included having a transaction ID of the correct length in hex format, a valid M, M input UTXOs in the correct format, N, and N vouts in the correct format, with semicolons, parens, and commas in the right places. For output, each comma and each semicolon shall be followed by a space, and the line shall end with a newline immediately following the last vout. There shall be no spaces between the end parens of one pair and the start parens of the next pair, or before a semicolon that ends a field. For input transactions, white space shall not matter as long as the whole transaction is on one line. If a transaction is not well-formed, an error message shall be output stating that the transaction is mal-formed (and optionally, why).

Each well-formed transaction shall then be checked for additional semantic correctness – each of the input UTXOs must appear as an output of a transaction already present in the ledger, and must not appear as an input to another transaction that is already present in the ledger (i.e., only previous UTXOs can appear as inputs to a transaction). If any input is not a valid UTXO, then an error message shall be output stating that the input is invalid (and optionally, why).

Each well-formed transaction with valid inputs shall also be checked for valid outputs, with the total of the amounts in the outputs summing up to the sum of the values of the UTXOs listed as inputs. If the sums to not match, then an error message shall be output stating that the outputs are invalid (and optionally giving details).

Example Run: (interactive)

```
$ ./ledger
[F]ile
[T]ransaction
[P]rint
[H]elp
[D]ump
[W]ipe
[I]nteractive
[V]erbose
[B]alance
[E]xit
Select a command: T
Enter Transaction: 84dfb9b3; 1; (40671f57, 0); 2; (Bob, 100)(Gopesh, 5)
Sorry, invalid transaction. Not enough money…
Select a command: B
Enter User: Alice
Alice has 850
Select a command: E
Good-bye
$
```

Grads Only:

Use SHA-1 to form TxID from the string consisting of the canonically formatted transaction contents, starting with the first character following the first semicolon and space. For a provided transaction, check that the transaction ID is correct. Use the transaction IDs given in the example ledger to test your results.

For example, from the command line on storm:

```
% cat trans1.txt
0; ; 1; (Alice, 1000)
% shasum trans1.txt
f2cea5395b48363f9b86fbd6c80ce83a70d46a51  trans1.txt
%
```

Take first 8 characters (first 4 bytes) as the identifier for the first transaction.

Check the TxID provided for each provided transaction and correct it with a suitable error message if the TxID is wrong.

Next Exercise: We will be adding signatures with pub/priv key pairs and validate the signature too. Base64 will be used for address for the keys.

Submission: (use C, C++, or Java)

1) a makefile for all programs and support code that makes the executable; make all must do whatever is necessary to create an executable file called ledger that will run the program;

2) all source code (header files, etc.);
3) README.txt file that explains the environmental expectations of the code and lists any bugs and how to run it;
4) Brief textual report in text, doc, docx, or PDF format that reflects on how you approached the problem and how you solved any challenges, what you did to test the program, and what you learned from the assignment.

Compile errors will result in zero points as non-testable. Segmentation faults and core dumps will be judged depending on how much correct information is displayed. TEST YOUR MAKEFILE AND CODE ON STORM BEFORE SUBMITTING!

Rubric: (all) 15 points possible

5 pts - Input filename and parse ledger with correct output
3 pts – Input a single transaction
1 pts - [P/p] functionality
1 pts - [B/b] functionality
1 pts - [W/w] functionality
1 pts - [H/h] functionality
2 pts - Textual Report – esp. testing
1 pts - Messages to stderr

(grad) 5 additional pts possible
5 pts – correct assignment and checking of SHA-1 hash transaction IDs. Additional point for report.