Ledger Loops: Debt loops across local ledgers (Work in Progress)

Michiel B. de Jong

November 2016

Abstract

LedgerLoops is a protocol for clearing debts and credits across multiple ledgers. Unlike Bitcoin, which introduces one global public ledger, LedgerLoops acts on a loop of two or more local ledgers. This whitepaper introduces the concept of second-order debts which is at the core of LedgerLoops, and explains how they can be created using the two LedgerLoops contract types. It also defines a messaging protocol which can be used to send these contracts along the ledger loop in a peer-to-peer fashion, and a decentralized algorithm to find cycles in a debt graph.

1 Second-order Debts

A debt can be represented by an "I owe you asset A"-message ("IOU" for short), sent from the debtor to the creditor. This would be a first-order debt. A second-order debt would take the form "I owe you asset A from the moment person X owes asset B to person Y". It doesn't matter who these persons Xand Y are (they might be complete strangers), nor what this asset B is (its description might be a reference only persons X and Y understand).

A second-order debt does not immediately give the receiver a claim to the asset that is owed by the sender; however, if the receiver can present cryptographic proof that at any point in time, person X owed asset B to person Y, this triggers the sender's debt into action. So a second-order debt, combined with a cryptographic proof that the debt it refers to exists, or has existed in the past, is equivalent to a first-order debt. We say the cryptographic proof of the statement "Person X owes asset B to person Y" *activates* all second-order promises that are contingent on this clause.

Settling a second-order debt (after it has been activated) works the same way as settling a standard IOU (a first-order debt): the receiver of the IOU sends a message back to the IOU-sender, stating that the IOU-sender no longer owes the assets mentioned in that specific debt statement.

2 Contract Type I

Contract Type I is used to define first-order debts on which second-order debts can be based. It is written down as a canonical-JSON document containing the following data:

```
{
  protocolVersion: 'ledgerloops-0.4',
  msgType: 'contract-type-i',
  keyAlgorithm: <either 'ed25519' or 'rsa-256'>, // TODO: find out
      if these strings fully but minimally define how to interpret
      pubkey/pubkey2
  pubkey: <a public key for which person X holds the private key,
      unique to this contract>,
  pubkey2: <a public key for which person Y holds the private key,
      unique to this contract>,
  currency: <a three-letter ISO-4217 code, for instance 'USD'>
  amount: <a positive, terminating decimal number, for instance
      '13.5264235'>
}
```

3 Contract Type II

Contract Type II is used to define second-order debts. A Type II contract refers to exactly one Type I contract. It does so by mentioning the 'pubkey' field from the Type I contract in question. Type II contracts are written down in almost the same a canonical-JSON document, only the 'msgType' field is different:

```
{
  protocolVersion: 'ledgerloops-0.4',
  msgType: 'contract-type-ii',
  keyAlgorithm: <either 'ed25519' or 'rsa-256'>, // TODO: find out
      if these strings fully but minimally define how to interpret
      pubkey/pubkey2
  pubkey: <the public key of person X>,
  pubkey2: <the public key of person Y>,
  currency: <a three-letter ISO-4217 code, for instance 'USD'>
  amount: <a positive, terminating decimal number, for instance
      '13.5264235'>
}
```

In the current version of LedgerLoops (version 0.4), the asset mentioned in Type II contract must be the same as the asset mentioned in the Type II contract it refers to, therefore also exposing the nature of the asset from the Type I contract to the participants in the Type II contract. A further

restriction is that the asset consists of an amount of money, expressed in an ISO-4217 currency.

Lifting these restrictions makes the algorithm a bit more complicated, but is definitely on the roadmap for future versions of LedgerLoops.

4 The Algorithm

To get a feel for how the LedgerLoops algorithm works, consider the following minimal example, involving three ledgers:

/beginitemize John and William maintain a peer-to-peer ledger, on which John owes William one US dollar. William and Peter maintain a peer-to-peer ledger, on which William owes Peter one US dollar. Peter and John maintain a peer-to-peer ledger, on which Peter owes John one US dollar. /enditemize

John, William, and Peter can each only see two of these three ledgers, so none of them know that a debt cycle John -¿ William -¿ Peter -¿ back to John exists. They find this out and settle all three debts, they send the following messages:

4.1 Round one (debtor to creditor)

John, William and Peter all send the following message to William, Peter and John, respectively: "I have you as a creditor, but I also have at least one debtor, who may have other debtors, etc., eventually leading back to you."

4.2 Round two (debtor to creditor)

John, Peter and William all send the following message to Peter, William and John, respectively: "Right, so you said you have at least one debtor, try sending them this token, instructing them to forward it, so I can see if it comes back to me." (accompanied by a long random string which is generated by John).

4.3 Round three (debtor to creditor)

John sends the following message to Peter: "Right, the token came back so that's promising. Here is a public key which I just generated, you may want to use it in the pubkey2 field of a Type I contract."

Peter also generates a public key, for use in the pubkey field of a Type I contract which he may create later. For now, he only creates a Type II contract for 1 USD, reserves these funds on his ledger with William (i.e., will

not use this credit for anything else as long as William may claim the second-order debt from this Type II contract), and sends the Type II contract to William.

William sends a Type II contract with the same contents to John, after which John sends one to Peter.

4.4 Round four (creditor to debtor)

Peter notices that in the Type II contract he received from John, the pubkey is the one he himself generated (only Peter himself knows this) and the pubkey2 field is the one John sent him (both John and Peter know this). He now generates and signs the Type I contract which corresponds to all Type II contracts sent so far, and sends it back in the opposite direction - back to John - accompanied by a message which amounts to:

"Hey, look what I got - a valid signature for the Type I contract corresponding to the Type II contract you just sent me. So that successfully activates your second-order debt, which we can settle against my 1 USD debt on our peer-to-peer ledger."

John confirms to Peter that he agrees to cancel Peter's pre-existing debt against his own activated second-order debt.

Like Peter earlier, John now has the valid signature of the Type I contract, and can use that to activate William's second-order debt, and settle his debt with William.

After that, William now has the valid signature of the Type I contract, and can use that to activate Peter's second-order debt, and settle his debt with Peter.

All debts have now been settled, and the goal has been achieved. None of the participants know (except by looking at how quickly the messages looped round), how long the loop was in which they participated.

5 Conclusion

This is a work in progress. Full implementation details of LedgerLoops 0.4, as well as an initial discussion of security considerations, will be documented soon, once the code on https://github.com/michielbdejong/ledgerloops stabilizes a bit.