On-Chain Invoicing

powering Revolutionary.Cash

Version 1 – DRAFT 1

2021-01-18

Authors:
Bill McGonigle <bill@bfccomputing.com>
Bonnie Scott <bonnie@bfccomputing.com>

ABSTRACT

This document presents On-Chain Invoicing ("OCI"), a cryptographically-secure, censorship-resistant peer-to-peer messaging protocol that runs on top of the Cash chain of Bitcoin ("Bitcoin") for speed and economic efficiency. It provides the messaging-passing layer necessary to manage one-time and recurring invoices, payments, and management thereof without any single points of failure. Censorship is averted by decentralizing the finance model, using encrypted payloads, and encouraging users to connect using privacy-enhancing network traffic tools and wallets.

PARTIES

The typical usage model would be for a subscriber ("Sam") and a creator ("Charlie"). The protocol is peer-to-peer, so either party can have either role for each instantiation. That is, Charlie can be Sam's subscriber and Sam can be Charlie's subscriber, but only one role is appropriate per invoice. The terms "consumer" and "producer" or "customer" and "business" can be substituted in traditional business contexts.

USAGE MODEL

Charlie wishes to invoice for a product or service, possibly on a recurring basis. To do this, Charlie needs to specify what he wants (e.g. $5/month) and enter it into the Revolutionary Cash software (currently an Electron Cash plugin). With this, the software provides for him a Bitcoin address and a public key. Charlie will most likely give this to Sam in the form of an BCH/OCI QR-Code, of the form: `bitcoincash:bch_cashaddr?oci=pubkey_cashaddr` . Sam uses these two data elements to initiate the invoice negotiation with the Revolutionary Cash software.

MESSAGING

OCI uses several round trips, to establish an information-theoretical proof of agreement ("contract"), for similar reasons as the three-way handshake used by TCP, and also being sufficient for accounting purposes. Using fewer messages would require baked-in support for invoicing from a cryptocurrency or making certain assumptions about acceptance. Distributed financial solutions will always require some degree of verbosity to make up for the lack of a realtime connection. Use of zero-conf hints may accelerate the user experience, though the blocktime may not offer much impediment for the recurring subscription use cases (judicious use of notifications can help the user experience). On-chain time-based and escrow-based smart contracts offer a complimentary solution that may be preferable in some situations; OCI is not meant to displace them for their use cases.

MESSAGE FORMAT

Messages are carried in Bitcoin Cash OP_RETURN transactions. Each message is compressed with the Zstandard compressor (maximum level) and encrypted using a NaCl Box (EC25519 with 256-bit keys, a nonce and a checksum). The OP_RETURN data payload is prefixed with the little-endian bytes for 'REVO' (ASCII) for node filtering purposes, based on the Bitcoin Cash 4-byte prefix guidelines.

The data payload is built with FlexBuffers based on a schema that is fixed for all message types but containing data fields that are optionally populated, depending on message type definition. This minimizes data size and schema complexity. Tighter data packing than FlexBuffers will be explored before this document leaves DRAFT state.

Currently the maximum OP_RETURN data size is limited to 220 bytes, which is just barely enough for most secure invoicing messages. Memo fields may need to be limited to ensure an invoice will fit in one transaction. Multiple Bitcoin transactions or PUSHDATA's will be explored before this document leaves DRAFT state. Neither are desirable from a complexity or cost perspective. More parsimonious usage of existing bitspace is more desirable and the presumed direction this document will take.

MESSAGE TYPES

1: SUBSCRIBE

Generated by Sam, a SUBSCRIBE message is set to the predefined Bitcoin address, encrypted to the pre-shared public key, and Sam includes his public key in the message. Charlie's Bitcoin address is tied to the offer type in his Revolutionary Cash software. Every message to the same address will create the same type of invoice, so Charlie can run off a hundred copies of his code to hand out at a rally or performance.

2: PROPOSE

Charlie responds with a PROPOSE message which contains the full proposed invoice. Charlie's software makes an entry with the offer and a unique invoice id#. Charlie provides the payment address(es), at least one of which is a unique address for this invoice.

3: CONFIRM

Sam agrees to accept the invoice on his device, his software marks his invoice 'confirmed' and he sends a confirmation message to Charlie.

4: INVOICE

Charlie receives Sam's confirmation message, marks his side as 'confirmed' and issues a final invoice message which is sent to Sam. Sam will mark his side 'agreed'. This three-way handshake creates the virtual contract.

At this point Sam's software is responsible for paying the invoice on the dates and/or schedules specified.

5: MESSAGE

A MESSAGE message is sent from either party to the other party for comments on the invoice/contract. Software should timestamp and notify on these messages.

6: VOID

Either party may send a VOID message to cancel the contract/invoice. Software should carefully ask for confirmation.

DRAFT

CURRENT STATUS

The OCI software stack has been developed in Python3 on Linux and is currently functional as a command-line test scaffold. Performance on a Ryzen 5 desktop computer is under 30ms per message generated.

User documentation is under development at https://Revolutionary.Cash
A chart of International and Cryptocurrency codes for denomination of payments is at https://OCI.cash
A mailing list has been set up for discussion. See: https://SIG.cash Sign up for discussion and updates.

CONTINUING WORK

More work needs to be done to build a self-sustaining ecosystem. As of this DRAFT:

The Electron Cash plugin needs to be finished.
This document needs to be updated with data types and bitpacking standards, as finalized.
Source code needs to be bounds-checked, error-proofed, properly formatted, and released as a library. From there, code review and auditing.
The URI format needs to be finalized. Cashaddr format is great but OCI should use a unique type byte, at least, for its data representation.
On-boarding of deplatformed and at-risk creators.
Tor and Neutrino support is planned; I2C will be investigated.
Target marketing materials / outreach.
A mobile interface will be developed, independently or with wallet vendors.
Automatic “Revolutionary Recharges” need to be implemented in a wallet to purchase more Bitcoin Cash with fiat when wallet balances run low, for a smooth user experience.
A self-contained paper-wallet generator will be developed.
PDF generation for invoices will be added for legacy business needs.
Integration of fiat exchange-rate oracles.
Integration of web wallets (notably for “Usurious-Chat” services used by streamers).
Exploration of zero-conf possibilities.
Integration with WordPress/WooCommerce, then other e-commerce platforms.
Onboarding of traditional commerce with custom B2B integrations.
Whitelabel branding for corporate clients.
Website javascript ‘one liner’ library to integrate payments with websites. (“Zero-friction De-Fi”)
This document should up its game to academic standards (i.e. LaTeX)

FIGURES
1. QR-Code

(e.g. print n-Up on laser business cards)

DRAFT

2. Flow-Control Diagram

[this will be added shortly, when it’s better than it is now]

DRAFT