Light Clients for Building UTxO Ledger Transactions

Pyrros Chaidos$^{1[0000-1111-2222-3333]}$, Aggelos Kiayias$^{1[1111-2222-3333-4444]}$,
Marc Roeschlin$^{1[0000-1111-2222-3333]}$, and Polina
Vinogradova$^{1[0000-0003-3271-3841]}$

Input Output, Global `firstname.lastname@iohk.io` iohk.io

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

What sets us apart?

- Atomicity of payment+service
- Model for (trustless) 2-party transaction construction rather than proving things about chain/ledger state
- Do not require establishing a relationship with SP or any other set-up
- inherent timeliness of transaction construction incentivized by SPs desired to earn their tip. This is in contrast with the possibility of stale info provided from old Mithril snapshots in other LC models

We claim that our work can be used by any UTxO or EUTxO blockchain (with some adjustments to the details of intent specification). We use blind signatures [5]

2 Related Work

Compare our approach with :

- "Free" websites monitoring the chain – mention they lack long-term sustainability.
- Bridges (trustless and trusted)
- Payment channels
- LCs that operate on single-prover model (eg. with an established relationship via deposit)
- LCs that operate on multi-prover model
- LC SoK
- Solver networks

2.1 Technical Background

3 Ledger Model

The ledger model to which we tailor our light client design is a UTxO ledger with multi-asset support (UTxO_{ma}), first introduced in [3]. The UTxO ledger model, such as the one used by BitCoin [6], Ergo [4], and Cardano [1], maintains a record, called the *UTxO set*, of transaction outputs added by transactions that have been applied throughout its history, but not yet spent by subsequent transactions. For completeness, and in order to establish notation, we include an overview of the UTxO_{ma} model. For additional notation explanation, see Figure 2.

Multi-asset Support. A ledger which supports transacting with not only the primary currency, e.g. BitCoin on the BitCoin platform, or Ada on Cardano, but also other types of currencies, is called a *multi-asset ledger*. In this work, we chose to build on a multi-asset UTxO ledger model in order to demonstrate a broader range of usecases for our design. We do not, however, make use of the more expressive Extended UTxO ledger model [2]. We later discuss how our design can be adjusted to both the basic UTxO model, and the Extended UTxO with multi-assets model.

Each asset is uniquely identified by an $\mathsf{AssetID} := \mathsf{Policy} \times \mathsf{TokenName}$. Arbitrary combinations of assets are specified using a finitely-supported map $\mathsf{Value} := \mathsf{AssetID} \mapsto \mathsf{Quantity}$, where, for a given $v \in \mathsf{Value}$, all assets whose IDs are not included in the domain of v are assumed to have quantity $0 \in \mathbb{N}$. This data structure has a partial order $\leq$, and forms a group under addition $+$, with zero being the empty map $\emptyset$. In our model, all assets are user-defined, meaning that a user can introduce any asset into circulation so long as minting of this asset is allowed by its minting policy (which may be defined by the user themselves).

Ledger State and the UTxO Set. The ledger state is a data structure which is updated by applying incoming transactions. The state of a UTxO-based ledger necessarily contains a UTxO set. While realistic ledgers often contain additional information in their state, in our model, the ledger state is just the UTxO set. The UTxO set is a finite map, $\mathsf{UTxO} := \mathsf{TxIn} \mapsto \mathsf{TxOut}$. A transaction updates the UTxO set by either adding and removing entries.

Transactions. A transaction is the following data structure :

$$\begin{aligned}\mathsf{Tx} = (&\mathsf{inputs} : \mathsf{Set}\ \mathsf{TxIn},\\ &\mathsf{outputs} : [\mathsf{TxOut}],\\ &\mathsf{validityInterval} : \mathsf{Interval}[\mathsf{Slot}],\\ &\mathsf{mint} : \mathsf{Value},\\ &\mathsf{sigs} : \mathsf{Signature})\end{aligned}$$

An input $(txid, ix) \in \mathsf{TxIn} := \mathsf{TxId} \times \mathbb{N}$ is a pair of a transaction ID and a natural number. When a transaction is applied to a UTxO set, its set of inputs is used to identify the entries which the transaction is removing from the set. In each input, $txid \in \mathsf{TxId} := \mathbb{H}$ is the hash of a (previous) transaction that added that entry to the UTxO, and ix is the index of that output in the list of outputs of that transaction.

An output $(s, v) \in \mathsf{TxOut} := \mathsf{Script} \times \mathsf{Value}$ is a pair of a script s which specifies some constraints that are checked when the output is spent, and the assets v contained in the output. The list $\mathsf{outputs}$ of outputs of a transaction tx is used to construct a set of UTxO entries that will be added to the UTxO set, such that the unique identifier $txin$ of each output $o \in \mathsf{outputs}\ tx$ consists of the transaction hash $\mathsf{txid}\ tx$, and the index of o in the list $\mathsf{outputs}\ tx$. The entires added to the UTxO set by tx are computed in this way by mkOuts, see Figure 3.

A slot number $s \in \mathsf{Slot}$ represents blockchain time, and is specified at the block level, but we do not model the details of block application in this work. The interval $\mathsf{validityInterval}$ specifies the range of slot numbers for which a transaction can be valid. The field $\mathsf{sigs} : \mathsf{Signature} := \mathsf{PubKey} \mapsto \mathbb{H}$ is a set of public keys, associated with their signatures on the the transaction (excluding sigs itself).

The mint field represents the assets being minted or burned by the transaction. Assets with positive quantities are said to be minted, while those with negative quantities are burned. When a transaction is applied, the constraints specified by every $p \in \mathsf{Policy} := \mathsf{Script}$ of each type of asset specified in this field are checked to make sure minting/burning of this type and quantity of asset is allowed.

Ledger State Update. The ledger state is updated by transaction application. Given a UTxO set $utxo$ and a transaction tx, the function $\mathsf{updateUTxO} : \mathsf{UTxO} \times \mathsf{Tx} \to \mathsf{UTxO}$ computes the updated UTxO set by adding and removing the appropriate entries :

$$\mathsf{updateUTxO}\ (utxo,\ tx)\ =\ \{\ i \mapsto o \in utxo\ \mid\ i \notin \mathsf{inputs}\ (tx)\ \} \cup \mathsf{mkOuts}(tx)$$

While an update to the UTxO set can be computed for any transaction, only transactions that are *valid* for a given set are allowed to perform an update to the ledger state. For a given $utxo$ set, a transaction tx is valid whenever the function $\mathsf{checkTx} : \mathsf{Slot} \times \mathsf{UTxO} \times \mathsf{Tx} \to \mathbb{B}$, applied as $\mathsf{checkTx}\ (utxo,\ tx)$, returns True. The $\mathsf{checkTx}$ function is the conjunction of the constraints specified in Section A.1. This includes checking that the constraints of every Script run by the transaction are satisfied. The constructors and evaluation of Script is given in Figure 1, and MOf is given in 3.

4 Light Client Specification

What is a light client?

- User - LC interface
- LC characteristics/ limitations
  - What are the capabilities of a light client?
  - Does it remember all addresses it has been paid at (tx history)?

Constructors of Script

$$\begin{aligned}
\mathsf{RequireMOf} &: \mathbb{N} \to [\mathsf{Script}] \to \mathsf{Script}\\
\mathsf{RequireSig} &: \mathsf{PubKey} \to \mathsf{Script}\\
\mathsf{RequireTimeStart} &: \mathsf{Slot} \to \mathsf{Script}\\
\mathsf{RequireTimeExpire} &: \mathsf{Slot} \to \mathsf{Script}
\end{aligned}$$

Evaluation of Script

$$\begin{aligned}
[\![\_]\!] &: \ \mathsf{Script} \to ((\mathsf{SetPubKey}) \times (Slot \times Slot)) \to \mathbb{B}\\
[\![\mathsf{RequireMOf}\ n\ ls]\!](khs, (t1, t2)) &= \mathsf{MOf}\ 0\ n\ [\![\_]\!]\ ls\\
[\![\mathsf{RequireSig}\ k]\!](khs, (t1, t2)) &= k\ \in\ khs\\
[\![\mathsf{RequireTimeStart}\ t1']\!](khs, (t1, t2)) &= t1'\ \leq\ t1\\
[\![\mathsf{RequireTimeExpire}\ t2']\!](khs, (t1, t2)) &= t2\ \leq\ t2'
\end{aligned}$$

Fig. 1: Script constructors and evaluation

  - Do we assume that viewing keys exist? Can we assume LC can generate first x addresses from its private key in a deterministic way?
  - Is light client allowed to maintain state, and what state can they maintain if so? Secret key is the minimum state.
  - Sanity test: if we dismiss all requirements we should recover a full client.
- LC - Full Node interactions
  - Protocols to support user requests
  - Security reqs (protecting integrity/ privacy/ SPO revenue?)

Can we describe the differences between light wallets, bridges and light nodes in our framework? (Probably yes).

How can we formalize the intent of a light client without revealing secret key?

Can we have viewing keys?

5 Threat Model

- Client finds out too much from SP answer and can submit tx without payment to SP (that's why the inputs must be hidden)
- SP lies to client about the UTxOs being spent by the tx, and tricks them into doing something they didn't want to do (that's why 0-knowledge proof that outputs were correctly specified)
- Suboptimal transactions possibility : Do we want to let users specify what they want optimized for in their intents (e.g. minimize fee or find best price on exchange offer)? Can we assume that competition across SPs will enable client to get a better response?

6 Intent Specification

Intent (DSL or predicate to describe intent of the client, ie what they want to do to the ledger state).

Give example of :

- intent to mint some tokens before a deadline
- intent to pay x from key k1 to k2 (ie script $\mathsf{RequireSig}\ k1$ to $\mathsf{RequireSig}\ k2$)

7 Light Client Protocols

8 Analysis

9 Conclusion

Acknowledgments. A bold run-in heading in small font size at the end of the paper is used for general acknowledgments, for example: This study was funded by X (grant number Y).

Disclosure of Interests. It is now necessary to declare any competing interests or to specifically state that the authors have no competing interests. Please place the statement with a bold run-in heading in small font size beneath the (optional) acknowledgments[1], for example: The authors have no competing interests to declare that are relevant to the content of this article. Or: Author A has received research grants from Company W. Author B has received a speaker honorarium from Company X and owns stock in Company Y. Author C is a member of committee Z.

References

1. Cardano Team: Full Cardano Ledger. https://intersectmbo.github.io/formal-ledger-specifications/pdfs/cardano-ledger.pdf (2024)
2. Chakravarty, M.M.T., Chapman, J., MacKenzie, K., Melkonian, O., Müller, J., Jones, M.P., Vinogradova, P., Wadler, P.: Native custom tokens in the Extended UTXO model. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12478, pp. 89–111. Springer (2020). https://doi.org/10.1007/978-3-030-61467-6_7
3. Chakravarty, M.M.T., Chapman, J., MacKenzie, K., Melkonian, O., Müller, J., Jones, M.P., Vinogradova, P., Wadler, P., Zahnentferner, J.: $\mathrm{UTXO}_{\mathsf{ma}}$: UTXO with multi-asset support. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12478, pp. 112–130. Springer (2020). https://doi.org/10.1007/978-3-030-61467-6_9

[1] If EquinOCS, our proceedings submission system, is used, then the disclaimer can be provided directly in the system.

4. Ergo Team: Ergo: A Resilient Platform For Contractual Money. https://whitepaper.io/document/753/ergo-1-whitepaper (2019)
5. Fuchsbauer, G., Wolf, M.: Concurrently secure blind schnorr signatures. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. pp. 124–160. Springer Nature Switzerland, Cham (2024)
6. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/en/bitcoin-paper (October 2008)

A Appendix

Figure 2 introduces non-standard syntax we use throughout.

$$
\begin{array}{rlr}
\mathbb{H} = & \displaystyle\bigcup_{n=0}^{\infty} \{0,1\}^{8n} & \text{the type of bytestrings} \\
(a, b): & \mathsf{Interval}[A] & \text{intervals over a totally-ordered set } A \\
\mathit{Key} \mapsto \mathit{Value} \subseteq & \{\ k \mapsto v \ \mid\ k \in \mathit{Key},\ v \in \mathit{Value}\ \} & \text{finite map with unique keys} \\
[a1; ...; ak]: & [C] & \text{finite list with terms of type } C \\
h :: t: & [C] & \text{list with head } h \text{ and tail } t
\end{array}
$$

Fig. 2: Notation

Figure 3 lists the primitives and derived types that comprise the foundations of the EUTxO model, along with some ancillary definitions. (Outputs normally refer to transaction IDs by hash, but we simplify here for clarity.)

A.1 Transaction Validation Rules

(i) **The transaction has at least one input:**

$$tx.\mathsf{inputs} \neq \{\}$$

(ii) **The current slot is within transaction validity interval:**

$$\mathit{slot} \in tx.\mathsf{validityInterval}$$

(iii) **All outputs have positive values:**

$$\forall o \in tx.\mathsf{outputs},\ \ o.\mathsf{value} > \emptyset$$

(iv) **All output references of transaction inputs exist in the UTxO:**

$$tx.\mathsf{inputs} \subseteq \mathsf{dom}\ \mathit{utxo}$$

Ledger primitives

$$\mathsf{checkSig} : \mathsf{Tx} \rightarrow \mathsf{PubKey} \rightarrow \mathbb{H} \rightarrow \mathbb{B}$$

checks that a given key signed a transaction

Helper functions

$\mathsf{toMap} : \mathbb{N} \rightarrow [\mathsf{TxOut}] \rightarrow (\mathbb{N} \mapsto \mathsf{TxOut})$

$\mathsf{toMap}(\_,\ [\,]) \qquad\qquad = [\,]$

$\mathsf{toMap}(ix,\ u\ ::\ outs) = \{\ ix \mapsto u\ \} \cup \mathsf{toMap}(ix + 1,\ outs)$

constructs a map from a list of outputs

$\mathsf{mkOuts} : \mathsf{Tx} \rightarrow \mathsf{UTxO}$

$\mathsf{mkOuts}(tx) = \{\ (tx,\ ix) \mapsto o\ \mid\ (ix \mapsto o) \in\ \mathsf{toMap}(0,\ tx.\mathsf{outputs})\ \}$

constructs a UTxO set from a list of outputs of a given transaction

$\mathsf{MOf} : \mathbb{N} \rightarrow \mathbb{N} \rightarrow (A \rightarrow \mathbb{B}) \rightarrow [A] \rightarrow \mathbb{B}$

$\mathsf{MOf}\ k\ m\ f\ [\,] = m\ \leq\ k$

$\mathsf{MOf}\ k\ m\ f\ (h\ ::\ t) = \mathsf{if}\ (m\ \leq\ k)\ \mathsf{then}\ \mathsf{True}\ \mathsf{else}\ (\mathsf{MOf}\ (k\ +\ a)\ m\ f\ t)$

$\quad \mathbf{where} \quad a\ =\ \mathsf{if}\ (f\ (h))\ \mathsf{then}\ 1\ \mathsf{else}\ 0$

returns True *if enough elements of a list satisfy given function*

Fig. 3: Primitives and basic types for the UTxO_{ma} model

(v) **Value is preserved:**

$$tx.\mathsf{mint} + \sum_{i \in \ tx.\mathsf{inputs}, \ (i \ \mapsto \ o) \in \ utxo} o.\mathsf{value} = \sum_{o \in \ tx.\mathsf{outputs}} o.\mathsf{value}$$

(vii) **All inputs validate:**

$$\forall\, i \in tx.\mathsf{inputs},\ i \mapsto (s, v) \in utxo,\ [\![s]\!](\mathsf{dom}\ (tx.\mathsf{sigs}),\ tx.\mathsf{validityInterval}) = \mathsf{True}$$

(ix) **All minting scripts validate:**

$$\forall\, p \mapsto \_ \in tx.\mathsf{mint},\ [\![p]\!](\mathsf{dom}\ (tx.\mathsf{sigs}),\ tx.\mathsf{validityInterval}) = \mathsf{True}$$

(x) **All signatures are correct:**

$$\forall\ (pk \mapsto s) \in tx.\mathsf{sigs},\ \mathsf{checkSig}(tx, pk, s) = \mathsf{True}$$