Citrea (2) - Bitcoin L2

2026年の調査ページ。

2025年調査→Citrea - Bitcoin L2



https://docs.citrea.xyz/essentials/architecture-and-transaction-lifecycle

ChatGPT

Grok

- L2に関してはBitVM2ではないそうだ
  - Batch Prover が ZK proof して Bitcoinに書き込み、Full Nodeが検証する
    - 検証がNGでもBitVM2のchallengeなどはなく、Full Nodeがブロックを無視する
    - このバッチ処理の間隔は一定ではなさそう。
    - ブロックが作られる前の状態は Soft Confirmation と呼んでいる。mempoolにある状態と同じ感じ。2秒間隔。
  - ブリッジのClementine v1だけBitVM2らしい

cBTCとctUSDは関係ない

Grok

Citreaのドキュメントに載っているトークンは**cBTC**と**ctUSD**だけである。

ガバナンストークンとして**CTR**というものが出てきたらしい。

USDC.e や USDT.e などはブリッジした資産で Citrea公式ではない。

cBTCはCitrea L2のネイティブトークンで、ClementineブリッジからPeg-InしたBTCと価値が1対1である。

zkEVMのガス代はこのcBTCを使うことになる。

decimals は 18 で、ここは EVM互換にしている。

ctUSDはステーブルトークンで、decimals は 6でUSDTやUSDCにならっている。

USD

表示例	実際の価値	説明
1 ctUSD	1.00 USD	1ドル
0.5 ctUSD	0.50 USD	50セント
0.01 ctUSD	0.01 USD	1セント
0.000001 ctUSD	0.000001 USD	最小単位（1マイクロドル）

cBTCの入手経路は以下がある。

- ClementineによるBTC–cBTCブリッジ(ただし 10BTC単位)
  - cBTCの供給量が変化するのはこの経路だけ
- Clementineによって供給されたcBTCを交換する
  - Citrea上のDEXでスワップ(ctUSD、USDC.e、USDT.e、wBTC.e などと cBTCを交換)
  - クロスチェーンブリッジ

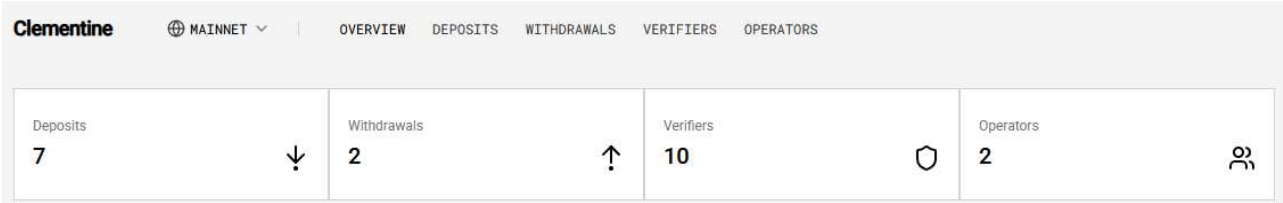
cBTC

ルート	新規cBTC発行？	実態	主な用途
Clementine	はい	BTC → 新規mint cBTC	一次供給（大口）
Symbiosis / Atomiq 等	いいえ	既存cBTCとのスワップ	小口・高速
DEXスワップ	いいえ	ユーザー間取引	即時取引
他チェーンブリッジ	いいえ	既存cBTCの移動	クロスチェーン

ブリッジ

Clementine Explorer

2026/05/06見たところでは、デポジットは7件、ウィズドローは2件。



ブリッジのClementine v1はBitVM2なのでPeg-In額は固定である。10BTCである。なので利用者は多くないだろう。

https://docs.citrea.xyz/developer-documentation/clementine-cli

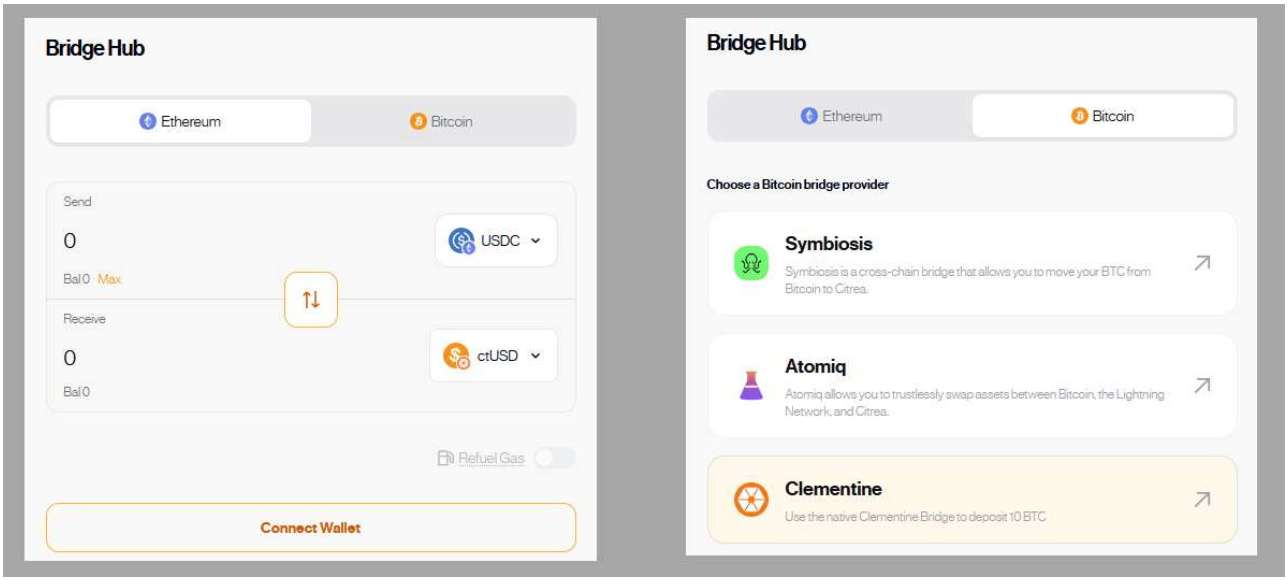
- TX例

Bridge Hub

https://citrea.xyz/bridge

ブリッジのページは Ethereum と Bitcoin に分かれている。”Clementine" は Clementine CLIのドキュメントページに飛ぶ。

現在は Clementine v1のはずである。

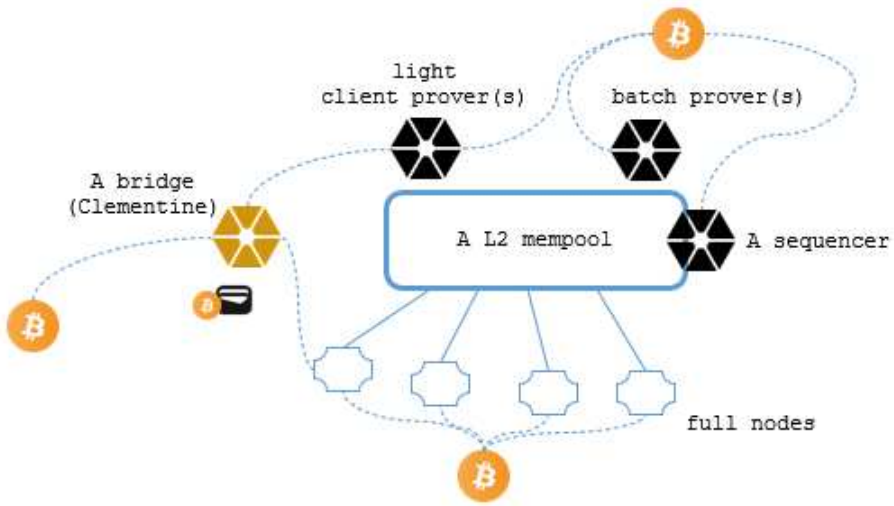


Symbiosis と Atomiq は 10BTC 単位でなくても cBTC にできるようだったので、これらは Citrea とのブリッジではなく Boltz のような交換かもしれない。

ブログに紹介してあるリポジトリは2026/04/28に更新している様子があったのでまだ開発は続いているのか？

https://github.com/chainwayxyz/garbled-snark-verifier/commit/cc513450c996f23e99e8eb732ce725338af53670

構成

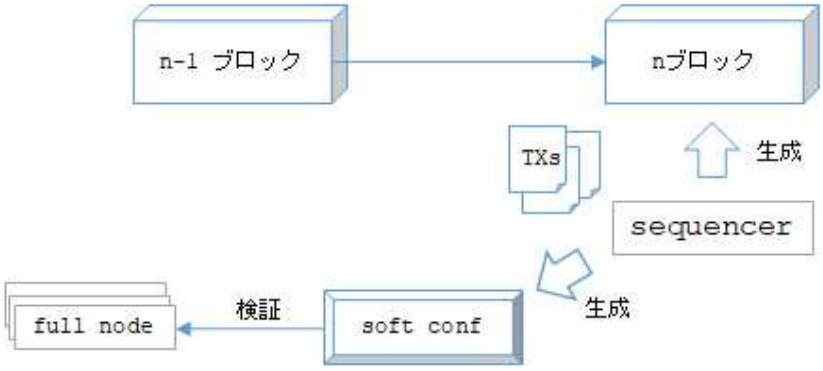


Sequencer

https://docs.citrea.xyz/essentials/architecture-and-transaction-lifecycle#sequencer

Grok

- sequencerは1つ
- トランザクション順序決定
- Rollupブロック生成
  - mempoolを2秒ごとに監視し、feeで優先度付して有効トランザクションのバッチを選択してブロックを作る
  - 署名
  - **soft confirmation**を作って展開
    - 新しいブロックのためにトランザクションの順番や状態変化を表すデータ(state root)を作り、そのブロックと前ブロックに関してハッシュ値やら何やらをまとめてsequencerが署名する
    - まだZK証明はないしBitcoinにも展開しない
    - 検証は Full node
      - Full nodeは新しいブロックのデータと soft confirmationのデータを検証
      - 問題なければ、そのfull nodeは状態を更新する
      - 問題があれば、そのfull nodeは状態を更新しない



Bitcoinと同じく誰かが拒絶したからといって全体が止まるわけではない。

Sequencer Commitmentを生成してBitcoin L1にトランザクション展開

Sequencer Commitmentに含めるブロックの範囲を決める(固定ブロック数？)

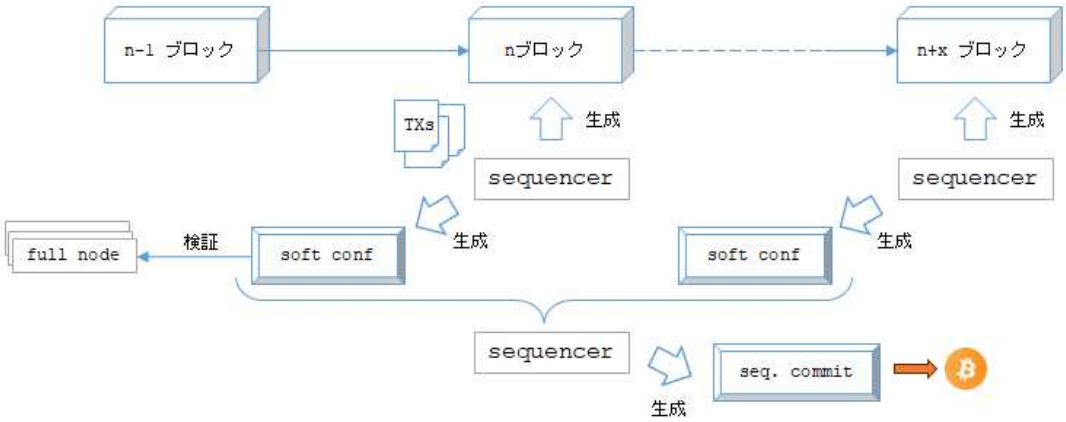
ある区間のsoft confirmationをそれぞれハッシュにしてマークルツリーを作り、マークルルートを得る

それをトランザクションデータにして展開

Sequencer CommitmentのTX例

Sequencer Commitment Index	L2 Start Height	L2 End Height	Status	Commitment	Commitment Transaction ID	Proof Transaction ID
6965	6964001	6965000	Committed	8818c..c9999	dab59..dbcfa	N/A
6964	6963001	6964000	Proven	d1fb1..08d85	9d5c9..4e80f	64434..d87ef

"Sequencer Commitment Index 6964" の Commitment Transaction ID



batch proverはSequencer Commitmentを使ってZK Proofを生成する

Sequencer Commitmentは順番に関して使うだけで他のデータも使う

Mempool

https://docs.citrea.xyz/essentials/architecture-and-transaction-lifecycle#mempool

mempoolは1つでSequencerが持つ(なので private mempool と呼んでいる)

Full NodeのRPCで展開されたとしてもSequencerが受け取って有効性チェックが行われてようやくmempoolに格納される

Full Nodeはmempoolを持たない。しかし soft confirmationしたブロックデータは持っているので、RPCでの問い合わせに対してはローカルのDBから値を返すことができる。

RPCで "pending" なデータは返せないだろう

rethのtransaction-poolを使っているようだ

Rust Ethereumなので reth(Go Ethereum だと geth)

Batch Prover

https://docs.citrea.xyz/essentials/architecture-and-transaction-lifecycle#batch-prover

Grok

複数運用可能

現状はRISC Zeroを使っている：https://docs.citrea.xyz/future-research/multi-prover

バッチ単位のZK proofを生成してBitcoin L1にトランザクション展開

前回の分から、L1に展開したSequencer Commitmentの順番でCitreaブロックを実行するとSequencerが主張する新しいstate rootになる、ということの証明

Sequencer Commitment TXが十分にconfirmするまで待ってProofを作り始める(それぞれStatusがあるようだ)

Sequencer Commitment Index	L2 Start Height	L2 End Height	Status	Commitment	Commitment Transaction ID	Proof Transaction ID
6967	6966001	6967000	Committed	4f665..b34cb	6d0c1..d5c03	N/A
6966	6965001	6966000	Committed	749c3..3c28f	11203..f971d	N/A
6965	6964001	6965000	Committed Finalized	8818c..c9999	dab59..dbcfa	N/A
6964	6963001	6964000	Proven	d1fb1..08d85	9d5c9..4e80f	64434..d87ef
6963	6962001	6963000	Proven	fe86c..6cf2b	1391a..55573	e78a8..7b628
6962	6961001	6962000	Proven Finalized	f7537..bed7f	5dd7f..9ae59	1e92c..bd8cc
6961	6960001	6961000	Proven Finalized	5e0a1..5149e	24db4..123bc	5f28b..0454e

ProofのTX例

"Sequencer Commitment Index 6964" の のProof Transaction ID

Sequencer Commitment Index	L2 Start Height	L2 End Height	Status	Commitment	Commitment Transaction ID	Proof Transaction ID
6965	6964001	6965000	Committed	8818c..c9999	dab59..dbcfa	N/A
6964	6963001	6964000	Proven	d1fb1..08d85	9d5c9..4e80f	64434..d87ef

TXサイズが大きいが、1TXあたり400KB以内にするという機構があって分割することもあるのだとか。

テンプレートとしてRISC ZeroとSP1(Succinct Labs)の2種類がある

複数種類にすることで、片方にバグなどがあったときにも対応しやすいなどセキュリティの強化

同じ種類を複数台運用するとスループットが上がったり冗長性が増すなどの利点がある

Groth16回路

Full nodeはL1に展開されたProofを検証する

soft confirmation で得られるのが soft finality という弱い確定で、Proof を検証して得られるのが最終的なfinality

Proof TXが十分にconfirmするまで待って検証する

Light Client Prover

https://docs.citrea.xyz/essentials/architecture-and-transaction-lifecycle#light-client-prover

Grok

"Full Node"に対する"Light Client Node"のような意味合い

BitcoinのLight Clientがブロックヘッダとmerkle proofを使うなどして軽量検証しているイメージらしい

これ自体がLight Client Nodeになるわけではなく、Light Client Node向けのproofを作る

主にClementineで参照する

Light Client Nodeが参照してもよいそうだが、現在はFull Nodeのような汎用にはなっていない

複数運用可能

現在は1つで、将来的にもそうなりそう

L2全体の状態を含めた再帰的ZK proofを生成

再帰的：前回の証明が正しい＋今回の証明、のように前の照明の検証も回路に組み込んでいる

ここでいえば、genesis blockからの検証が積み上げられている

証明サイズと検証コストがほぼ一定なのに過去から現在までの履歴をカバーできるのがよい

これはBitVMの仕様ではなくCitreaの設計によるもの

Full Node

https://docs.citrea.xyz/essentials/architecture-and-transaction-lifecycle#full-nodes

Grok

複数運用が普通で誰でも立てられる

いわゆるフルノード的なもので、検証とデータ提供のみ行う

Citreaブロックの同期

DA(Bitcoinチェーン)の監視

Sequencer Commitmentの検証

Batch Proverが生成したZK証明の検証

RPCの提供

データの提供

Sequencer、Batch Prover、Light Client Proverのベースになるノード

＋トランザクションの順番付け、Soft Confirmation作成、Sequence Commitment作成＝Sequencer

＋バッチ単位でのZK証明作成＝Batch Prover

＋再帰ZK証明作成＝Light Client Prover

Soft Confirmationについては受け取って保持するだけでZK検証などはしない

Sequencer Commitmentの検証はするが、不正があってもブロックを受け付けないだけで

BitVM2のchallengeはしない(L2はBitVMではない)

Clementine Bridge

https://docs.citrea.xyz/essentials/architecture-and-transaction-lifecycle#clementine

Clementineは1つ

BitVM2の指示役といったところ

L2については full node経由で参照するだけ

light client proverが作ったzk proofを参照する

`clementine-cli` がwalletを持っているので、トランザクション作成もclementineがやるだろう。

基本的なライフサイクル

mempoolへのトランザクション展開

cBTCの送金

スマートコントラクト呼び出し(トークンの送金なども)

Full NodeのRPCを介して展開される。トランザクションはSequencerまで伝搬されて有効であればmempoolに格納される

ドキュメント

ドキュメントがどのくらい更新されているかわからないが当てにできるのがこれしかない。

https://docs.citrea.xyz/

開発

testnet4での環境もあるのだが、WSL2のHDD譲渡はいえ Citreの同期に1週間か2週間くらいかかった。

Clementine v1でのPeg-Inは 10BTC 必要なのもあり、試すくらいであればregtestの方がやりやすいだろう。

Citrea - Bitcoin L2#14199d

Citrea

https://github.com/chainwayxyz/citrea/blob/nightly/docs/run-dev.md
このリポジトリではシーケンサー、フルノード、prover(batch, light client)を担当するバイナリを作る。
フルノード以外は1つだけという仕様らしい。

> Batch Proverが「中身（取引）」を証明し、Light Client Proverが「繋がり（履歴とビットコインとの整合性）」を証明します

だそうだ。

commit-id : `0f40b48efbb161820bda3cb7fc0a10d3bd0d686c`

setup

最後の `make install-dev-tools` に時間がかかる。thumbv6m-none-eabiは何に使うんだ？ `crates/sovereign-sdk/module-ssytem/sov-modules-core/Makefile` で使っているようだ。
Ubuntuのデフォルトシェルがdashのため `source` が使えないらしい(コマンドラインではbashになっているので `source` が使える)。 `make install-dev-tools SHELL=/bin/bash` と指定して実行するのが良さそうだ。

makefile

```
To get started:
1. Restart your shell or run:
   source "/home/ueno/.bashrc"
2. Verify the installation:
   rzup --help


/bin/sh: 2: source: not found
/bin/sh: 3: source: not found
/bin/sh: 4: rzup: not found
make[1]: *** [Makefile:73: install-risc0] Error 127
......
```

ビルド

```
make build
```

かなり時間がかかる。1時間はかからないがそのくらいの余裕を持ってやろう。
生成される実行ファイルは `citrea` と `citrea-cli` のみ。

```
$ cargo install --locked dprint
```

L2だけのモード - MockDA

Run on Mock DA

Bitcoin Regtestすら使わない環境。 `--da-layer mock` を付けてシーケンサーを起動する。

今回はスルーしておく

Regtestモード

Run on Bitcoin Regtest

Bitcoinチェーンを使う環境では `sequencer_rollup_config.toml` をBitcoin RegtestのRPCに合わせて変更する。

sequencer_rollup_config.toml

```
node_url = "http://127.0.0.1:18443"
node_username = "testuser"
node_password = "testpass"
```

bitcoind を直接参照するのか。

`bitcoin-cli createwallet` で `citreatesting` という名前のウォレットを作成するようになっている。

`citrea` コマンドの実行は大きくsequencer、full node、batch prover、light client proverの4つがあるようだ。

`--rollup-config-path` のTOMLファイルはモードごとにTOMLファイルが異なる。 `[rpc].bind_port` がCLIの待ち受けポート番号になる。

full nodeとbatch proverは `[runner]` でsequencerのRPCを指すようにしてある。URLだけしかないがセキュリティ的なものはないのだろうか？

`--sequencer` オプションを使うとsequencerモード。

citrea sequencer

```
./target/debug/citrea \
  --dev \
  --da-layer bitcoin \
  --rollup-config-path resources/configs/bitcoin-regtest/
sequencer_rollup_config.toml \
  --sequencer resources/configs/bitcoin-regtest/sequencer_config.toml \
  --genesis-paths resources/genesis/bitcoin-regtest/
```

`--sequencer` オプションがないとfull nodeモードのようだ。これが標準形？で、これにオプ

ションを付けるとそれぞれのモードになるというところか。

citrea fullnode

```
./target/debug/citrea \
  --dev \
  --da-layer bitcoin \
  --rollup-config-path resources/configs/bitcoin-regtest/rollup_config.toml \
  --genesis-paths resources/genesis/bitcoin-regtest/
```

`--batch-prover` オプションをつけるとbatch proverモードになる。

citrea batch prover

```
PARALLEL_PROOF_LIMIT=1 ./target/debug/citrea \
  --dev \
  --da-layer bitcoin \
  --rollup-config-path resources/configs/bitcoin-regtest/
batch_prover_rollup_config.toml \
  --batch-prover resources/configs/bitcoin-regtest/batch_prover_config.toml \
  --genesis-paths resources/genesis/bitcoin-regtest
```

`--light-client-prover` オプションを付けるとlight client proverモードになる。

citrea light client prover

```
./target/debug/citrea \
  --dev \
  --da-layer bitcoin \
  --rollup-config-path resources/configs/bitcoin-regtest/
light_client_prover_rollup_config.toml \
  --light-client-prover resources/configs/bitcoin-regtest/
light_client_prover_config.toml \
  --genesis-paths resources/genesis/bitcoin-regtest
```

run.sh

```
#!/bin/bash

PARALLEL_PROOF_LIMIT="0"
ROLLUP_CONF="sequencer_rollup_config.toml"
OPTION="--sequencer resources/configs/bitcoin-regtest/sequencer_config.toml"
if [[ $# -eq 1 ]]; then
        if [[ "$1" == "sequencer" ]]; then
                echo "sequencer mode"
```

```
        elif [[ "$1" == "fullnode" ]]; then
                echo "full-node mode"
                ROLLUP_CONF='rollup_config.toml'
                OPTION=""
        elif [[ "$1" == "prover" ]]; then
                echo "batch-prover mode"
                PARALLEL_PROOF_LIMIT="1"
                ROLLUP_CONF='batch_prover_rollup_config.toml'
                OPTION='--batch-prover resources/configs/bitcoin-regtest/
batch_prover_config.toml'
        elif [[ "$1" == "lcprover" ]]; then
                echo "light-client-prover mode"
                ROLLUP_CONF='light_client_prover_rollup_config.toml'
                OPTION='--light-client-prover resources/configs/bitcoin-regtest/
light_client_prover_config.toml'
        else
                echo "unknown command: $1"
                exit 1
        fi
fi

PARALLEL_PROOF_LIMIT=$PARALLEL_PROOF_LIMIT ./target/debug/citrea \
        --dev \
        --da-layer bitcoin \
        --rollup-config-path resources/configs/bitcoin-regtest/$ROLLUP_CONF \
        $OPTION \
        --genesis-paths resources/genesis/bitcoin-regtest/
```

regtestのチェーンを作り直したら `citrea` のデータも消してやり直すこと(エラーログで気づくだろう)。

```
$ make clean-node
```

動かしているだけで毎秒INFOログが出力される。何も操作しなくてもERRORやWARNのログが出ていた。

ブロックを作ったときのログ

bitcoin側で `generate` したときのログ。 `394d～` はブロックハッシュではなかった。

`generatetoaddress`

```
2026-05-01T03:07:12.340902Z  INFO citrea_sequencer::runner: New block #1473, Tx
count: #0
2026-05-01T03:07:13.260432Z  INFO citrea_sequencer::runner: New finalized L1 block
at height 107
2026-05-01T03:07:13.340010Z  INFO citrea_sequencer::runner: Saved short header
proof for block: 394d192ce410c29c30b1f5bffaea3b6d952daead0ac18aa83124a36ae339825d
2026-05-01T03:07:13.365150Z  INFO citrea_sequencer::runner: New block #1474, Tx
count: #1
```

しばしばエラーログが出る

しばしばというほどでもないかもしれないが、 `citrea` を立ち上げているだけなのにERRORログが出る。たしかにログの時間がおかしいのだが WSL2のせいか？ だいたい64秒ごとに出ているように見える。bitcoin側でブロック生成していても変わらない。
後日やると出なかった。よくわからない。

`error`

```
2026-05-01T02:40:38.427606Z  INFO citrea_sequencer::runner: New block #84, Tx
count: #0
2026-05-01T02:40:37.989487Z ERROR l2_block_rule_enforcer::hooks: error=Timestamp
should be greater
2026-05-01T02:40:37.989548Z ERROR l2_block_rule_enforcer::hooks: error=Timestamp
should be greater
2026-05-01T02:40:37.989588Z ERROR citrea_stf::hooks_impl: error=Timestamp should be
greater
2026-05-01T02:40:37.989603Z ERROR sov_modules_stf_blueprint::stf_blueprint: Failed
on `end_l2_block_hook`: TimestampShouldBeGreater
2026-05-01T02:40:37.989656Z ERROR citrea_sequencer::runner: Sequencer error:
Timestamp should
be greater
```

急にwarningログがでる

頻度はエラーログほどではないがWARNログが出る。定期的ではないが30分くらい？
こちらも後日やると出なかった。よくわからない。

`warning`

```
2026-05-01T04:20:18.304668Z  INFO CommitmentService:
citrea_sequencer::commitment::service: Sent commitment to DA queue. L2 range:
#5001-6000, index: 6
```

```
2026-05-01T04:20:19.291488Z  INFO citrea_sequencer::runner: New block #6003, Tx
count: #0
2026-05-01T04:20:20.291098Z  INFO citrea_sequencer::runner: New block #6004, Tx
count: #0
2026-05-01T04:20:21.209503Z  WARN bitcoin_da::helpers::builders::body_builders: Too
many iterations mining for sequencer commitment reveal tx prefix
2026-05-01T04:20:21.289818Z  INFO citrea_sequencer::runner: New block #6005, Tx
count: #0
2026-05-01T04:20:21.373490Z  WARN bitcoin_da::helpers::builders::body_builders: Too
many iterations mining for sequencer commitment reveal tx prefix
2026-05-01T04:20:21.553459Z  WARN bitcoin_da::helpers::builders::body_builders: Too
many iterations mining for sequencer commitment reveal tx prefix
2026-05-01T04:20:21.737518Z  WARN bitcoin_da::helpers::builders::body_builders: Too
many iterations mining for sequencer commitment reveal tx prefix
......
```

clementine

Clementineの方はドキュメントになっていた。

https://docs.citrea.xyz/developer-documentation/clementine-cli/advanced