Science of Blockchain 2025
Conference Article
Received: 2025-03-13

Chia Gaming: Real Time Games of Skill With Enforced Rules and no Trusted Third Party *

Bram Cohen
Chia, Inc.
bram@chia.net

Adam Kelly
Chia, Inc.
adam@chia.net

Art Yerkes
Chia, Inc.
a.yerkes@chia.net

Abstract

Playing a game with a counterparty online can be very risky if the game outcome has high stakes.

We present a design and implementation of a system that can reliably connect two anonymous parties, allow them to play a turn-based game with all moves verified either by pre-agreed on-chain code on in a State Channel, eventually declaring a winner. The system also has some provision for exceptional cases such as an unresponsive or a malicious or buggy opponent that sends invalid moves.

If the State Channel is open, the moves can proceed at a rate faster than the underlying blockchain can create blocks.

Keywords: Distributed Protocols, Gaming, Formal Proofs

I. Introduction

We give an explanation of an implementation of real-time gaming on top of Chia using state channels. It supports two-player, turn-based games. A game following this template enforces the game rules without requiring any trusted third party.

A. Background

1. State Channels vs. Payment Channels

Lightning network [1] is by far the largest user of channels to date. It is a payment channel network that supports real-time payments between two counterparties who do not have a preexisting direct relationship with each other by routing through intermediaries. In order to support this, it requires pre-funding of payment channels. It does not support full-blown state channels because Bitcoin Script cannot support the complex logic required by unroll coins. State Channels are much more complex to route.

Chia gaming does not have a static channel network. Each session is ephemeral, requiring one transaction at the start of a session to set up the channel and another one at the end to tear it down. If there is an issue in the middle of a session, the state channel is unrolled to chain, and any hands pending at that time have to play out on chain.

Routing of Chia state channels can be done, but has not been implemented yet. It has the benefit of not needing a new transaction on chain at the beginning of each session, but comes with the downsides of needing prepaid liquidity across an entire network and requires much more complex logic to implement than routed payments.

2. The coin set model vs. the UTXO model

The Chia on chain programming environment is a good fit for channels because it was designed from the ground up to support them, based on lessons learned from Bitcoin. The table below contains an overview of the relevant differences:

*Cite (APA): Cohen, B., Kelly, A., & Yerkes, A. (2025). Chia Gaming: Real Time Games of Skill. *The Science of Blockchain Conference 2025*

Coin Set vs. UTXO	
Bitcoin	Chia
UTXO model. Only UTXOs, their sizes, spend requirements, and birthdays are stored.	Coin set model. Only things stored are coins IDs, their sizes, spend requirements, and birthdays.
UTXOs are identified using hash of transaction and index	Coins are identified using parent id and puzzle hash and size
Scriptpubkeys are passed the transaction (which includes new output creation) and either fail or accept it	Puzzles are passed solutions and return conditions which include creation of new coins
Bitcoinscript only supports simple logic and can only support covenants via hash chaining Signatures are required to be tied to a specific coin	CLVM trivially supports covenants and can implement capabilities using shells around puzzles. Signatures can be tied to a specific coin, puzzle, or neither
Supports signature aggregation via multi-round protocol on top of secp256k1	Supports trivial signature aggregation using BLS

B. *Simple State Channels With Mini-Eltoo*

- Alice sends a partially fully funded transaction to make the state channel coin to Bob
- Bob adds the missing funds to that transaction, making it valid, then publishes it to the mempool. After the transaction goes through, the channel has been created and play proceeds
- The state channel coin is a standard format coin which implements 2-of-2 signing between Alice and Bob using BLS aggregation at the signature layer
- Alice and Bob alternate sending half-signed unrolls of updated state to each other. The state has a sequence number which is always increased. Alice always uses even numbers and Bob always uses odd numbers. The signatures include both an unroll of the channel coin to an unroll coin and an update to an unroll coin in case an obsolete version of it is put on chain
- Eventually either Alice and Bob agree to a clean shutdown and sign the channel coin going directly to their payouts or one of them countersigns a state from the other one and spends the channel coin to an unroll coin
- An unroll coin can be spent in one of two ways: After enough time has passed it can be spent to its specified state. At any time it can be spent by a signature being presented of a higher state number of the opposite parity. Because it's limited to two counterparties and uses parity both sides have then had a bite of the apple and there's no need to allow multiple recursive preemptions, hence 'mini-eltoo'.
- The final on chain state is generally a change coin for Alice, a change coin for Bob, and a game coin for each hand pending when the unroll happened

C. *Avoiding revealing/evaluating game state on chain with referee coins and slashing*

When played out on chain, a game is represented by a coin whose program contains the 'referee' program and the current state. Players take turns spending the coin, making a new state with each spend.. The game coin optimistically accepts any move but if an illegal move is played, that allows the opponent to slash.

Referee program state (simplified): Previous validation program hash Last move New validation program hash Money split if the next player concedes Current player's public key Other player's public key

Methods by which a game coin can be spent: Move Takes a new validation program hash, move, and split Must be signed by the current player's public key Makes a new referee coin with the specified new validation program hash, move, and split, the previous validation program hash set to this coin's new validation program hash, and the player keys swapped Timeout/concede Creates two payouts, one to each player, set to their public keys Requires a relative timelock but not signature This is how all games end which don't get slashed. Usually the last move specifies the split and doesn't allow a response. Slash Takes a previous validation program reveal and possibly evidence Runs the validation program passing it the new validation program hash If the validation program fails the transaction fails. Otherwise the player to move gets a payout of all the money.

For example, in a word game, we can prove that an opponent specified a word, W, that is not in the game dictionary. This can be proven by giving a range of words not in the dictionary and a signature of that range, signed by the key which set up the dictionary.

We need not validate earlier moves; these previous states are assumed to be entirely determined in the validation program. Slashing is the one and only time any of the actual game logic is performed on chain. This results in vastly reduced on chain cost in the common case.

D. System Overview

The system design, protocol design and division of code responsibility intend to keep the on-chain code small and simple, while allowing arbitrarily complex games. We do this by coding only the most vital rules of the game into the on-chain referee, coding the next most important rules in the verifier and handler layer, and the rest of the game in a conventional programming language. In our example games, we further separate the game client code into the game functions that need not be predetermined to play the game but still represent core game functions (like receiving messages from the blockchain, signing blockchain transactions, etc.), which are implemented in ChiaLisp, and functionality which generically belongs to the framework, which we have written in Rust. The Presentation and User Interaction layer is written in Typescript.

None of the game specific code is written in Rust. Everything associated with the game is loadable script code, whether or not it's essential to the on chain game state

1. System Parts

The validation program for a specific move is a program the referee uses to enforce rules for that particular turn of the game, and the referee identifies a reveal of the validation program for a specific turn by its hash.

Part A move handler takes the output of the move validation program and produces the two outputs, a convenient representation of the game for the presentation layer to use, and a "game state", which is essential information for the progression of the game. There is a one to many relationship

The handlers accept representations of user moves and evolve both the future handlers and the verified game state, producing reports to the user interface as a side effect

In general, the verifiers and handlers could be run, in an identical order, with identical inputs, and produce the same game, on or off-chain. The only asymmetry is on the zeroth move

2. Security benefits

The move verifiers and handlers could be written in any language, as they are executed off-chain, but in the case of CalPoker, we found it facilitated early development to write these in ChiaLisp, because of the ease of processing some data structures shared with the on-chain code.

E. System Benefits

Both Players can verify that the referee and handler code matches a publicly known hash expected for the chosen game.

- Both players (or their game client, or a third party) can prove that none of their core game code changes during the gaming session.

These are very high security guarantees compared to commercially available networked games.

F. System Limitations

A limitation of our System is that it currently only supports two players. We feel that this demonstrates the technique while allowing a wide variety of interesting games to be written.

The games must also be turn-based. Even though those turns can proceed at rate determined the round trip time of the network. Keep in mind that this lower-level, higher security game could be embedded in a larger conventional game, granting the benefits of of this system without the added development work of writing the entire game within this framework.

Another limitation is that a programmer new to cryptography or the programming language of the blockchain in question will have a bit of a learning curve in writing their first game using these features. We are attempting to make that process tractable by creating a framework that contains most code that will be commonly used in such an application, allowing users to focus on writing their game and presentation logic.

G. Designing a game for on chain play

The requirements of state channel gaming are very stringent:

There must be exactly two players: Even supporting secure channels with more than two users seems very difficult. The literature around envy-free cake splitting is not encouraging.

Turn based: No timing or video game type elements can be enforced

Very few turns: When an unroll to chain occurs game play can be very slow. The fewer turns there are in the game the less opportunity for griefing. This is also good for player engagement.

Small moves: Moves always need to be revealed on chain and so should be as small as possible, even though only the game logic has to be revealed in the event of a slash

Simple game logic, compatible with verifying on chain: When a slash does happen the verification program for that needs to be revealed and run on chain so the validation program must be cheap and fast to run on-chain.

Randomness can be achieved using commit and reveal: Simple randomness is easy to support this way but permutations/card removal effects are difficult. More on this in the explanation of the Poker implementation.

Chia gaming is rolling out with a suite of games showing the possibilities of the medium, how to do it, and exercising its edge cases:

1. California Poker: Similar to Chinese Poker, involves simultaneous play, the 'reference game'.
2. Krunk: Similar to Wordle but one player picks a word and the other tries to guess it. Requires use of a dictionary.
3. Space Poker: Similar to Holdem with rule changes to suit the medium. Has more complex recursive game flow than the others. Because players never make claims about their secret information which later get audited they get to keep their remaining stack when they fold early, unlike other games where premature folding has to result in getting nothing.

H. Example game: Poker

1. Implementing mental poker is a subject of much research. The approach taken here is as follows:
2. Alice and Bob choose their preimages. They both take turns committing to their 5th images, meaning hash(hash(hash(hash(hash(preimage)))))
3. Alice and Bob both reveal their 4th images. Alice's hole cards are found by hashing together her preimage and Bob's hole cards are found by hashing together his preimage and Alice's 4th image. If either player tries to lie about their 4th image the validation program will allow them to be slashed because the hash won't match the 5th image commitment.
4. There then is the preflop round of better which ends when either a player folds or both players call
5. Both players reveal their 3rd images which are hashed together to determine the flop cards
6. There is the flop round of betting
7. Both players reveal their 2nd images which are hashed together to find the turn card
8. There is the turn round of betting
9. Both players reveal their 1st images which are hashed together to find the river card
10. There is the river round of betting
11. Both players reveal their preimages to determine the winner

An obvious problem with this approach is that there can be a duplicate card, which will happen about half the time when choosing 9 cards randomly out of 52. The simplest approach to fixing this is to play a game variant which uses an infinite deck. For the purposes of simplicity this is the approach being taken initially. Sorry not sorry.

A more conventional fix which conforms exactly to the standard rules of Texas hold 'em is: Before the hand is even started have both players choose a 5th image then do a multi-party calculation to find out if there's a duplicate card and if so skip that hand. This is favored in many different ways. It has no impact whatsoever on on-chain play, where the costs of running programs on-chain need to be optimized. It can be done asynchronously far in advance. The form of multi-party calculation is only for two parties. It merely needs to be able to detect when one of the participants misbehaved without having to worry about any sensitive information being detected in the process because the hidden information isn't sensitive.

This much more favorable approach seems to not be common in the literature because it's centered around supporting all poker variants, which is more complex. But holdem is by far the most popular variant and the easiest to implement so it's completely legitimate to use its special properties to make implementation easier.

Several people have claimed that this is practical on normal hardware, but no implementation has been offered and the authors attempted to try benchmarks of available libraries and the results weren't encouraging.

II. Peer Discovery and Connection

Before two players can establish a State Channel, they must first find an interested peer. This is done by using a lightweight REST service.

A player, "Alice", offers a game by retrieving a unique, difficult to guess URL from the service which acts as a contact point. She then connects to a websocket specified by the service, and listens for messages. Offline she adds the game parameters she desires and a share secret and sends those vvia secure direct message of her choice to Bob. Bob receives this information, uses it to connect to a websocket associated with the shared URL, and sends the first peer handshake message. That message is repeated to Alice, and the peer connection is established. The message passing service can't read the messages because they're encrypted with the shared secret sent via a different channel.

III. TLA+ Model discussion

TLA+ [2] is a formal proof system for modeling programs and systems, especially concurrent and distributed systems. The authors make use of this tool to prove that there are no states that cannot reach the "end game" state. Note that this TLA+ code only models the peer connection between state channels, abstracting away anything

except the state of the peer connection.

The authors used a TLA+ program that models the operation of the peer to peer protocol. It can detect the possibility of state of the system in which one of the parties has an incorrect idea of who has control of the State Channel. This process was very helpful in verifying that our channel state transition code was valid, no matter the timing of sent or received messages. Specifically, the model aims to prove that given the initial conditions

```
Init ==
  /\ a = PotatoHandler(HandshakeStepA)
  /\ b = PotatoHandler(HandshakeStepB)
  /\ ui_actions = UIActions
```

we can always reach "Completed" state in both clients. That is, the system is deadlock free at the State Channel protocol level.

Using our model, we found and fixed a bug during development and testing. The model was also useful in another way: after reviewing the protocol model, it was possible to simplify the protocol, even though not all authors were as experienced with the implementation language at the time.

The State Channel Handshake & Teardown protocol is a token ring protocol between the two participants that establishes and enforces an order on the State Channel messages between the participants Alice and Bob. The protocol is conducted off-chain, although there are a few interactions between the blockchain and the protocol. For instance, the protocol receives an event when the State Channel coin is created.

Our TLA+ program models the non-blockchain code (implemented in Rust in our example) that implements the peer wire protocol that determines who has control of the State Channel at any given time. In the parlance of this particular Handshake and Teardown implementation, the party that is allowed to send the next State Channel message has "control" of the channel, or equivalently, "has the potato".

The model does not include messages from the particular game or any other communication over the state channel. From the point of view of this model, all application-layer communication happens during the "Finished" state.

Acknowledgement

We thank Dan Boneh for his advice and Leslie Lamport for creating TLA+.

Authors' Contributions

All authors participated in drafting the manuscript. All authors read and approved the final version of the manuscript.

Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability

All code discussed is available at github.com/Chia-Network/chia-gaming

Ethical Statement

In this article, the principles of scientific research and publication ethics were followed. No AGIs were harmed (or used) in this research, or the creation of this document.

References

[1] Poon, J., & Dryja, T. (2016). The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. lightning.network/lightning-network-paper.pdf

[2] Lamport, L. TLA+ Website lamport.azurewebsites.net/tla/tla.html

[3] Yerkes, A. (2025) TLA+ quickie: modelling rust code prozacchiwawa.medium.com/tla-quickie-modelling-rust-code-b2fe432fa25b