We could leverage commitment squashing and private kernels to create efficient state channel scheme. Here is how it could work for turn-based games:

1. Users A and B initialize the game by depositing a stake in the game contract and passing in identity_hash

.

1. User A calls turn(0) function where 0 is a turn_counter. Even number corresponds to user A, odd to user B.
2. turn(counter2)

function then does the following:

3.1 Calls get_secret

oracle and checks that hash(secret)

is equal to identity_hash

of user A if counter % 2 = 0

or user B if counter % 2 = 1

,

3.2 calls get_turn

oracle and progresses the game state,

3.3 calls turn(counter + 1)

if the game has not reached final condition. If final condition was reached finalize()

function gets called and the user submits the result for settlement.

1. If the game did not finish in point 3.3, user A submits the proof to user B and user B continues with the game (point 3.1)
2. Loop until finalize()

function gets called.

Efficiency

Since the game state would be represented by a note (or a set of notes) and we would nullify and “re-create” this game state note in each turn the commitments would get squashed and only the final note would be posted on chain.

Issues

This scheme is elegant but what if

1. I start losing and I decide to not continue with the game?
2. I decide to revert the game a few turns?

Joe’s solution to problem 1

Context

: “I am a user A and user B decides to grief the game and never send user A his turn.”

User A calls slash(userB)

function on the game contract. Given that user B might be completely innocent and user A might actually be the malicious one and ignoring user B’s response, user B will have the power to revoke the slashing by calling revoke_slash(kernel_proof, new_turn_of_user_B, secret)

function. This function will verify the proof, revokes the slashing and progresses the game state. If user B never calls revoke_slash

the game can be finalized by user A, making him a winner.

If one of the players decides to be malicious again the game will progress on L2 via the slash

revoke_slash

functions.

The worst outcome the attacker can achieve is forcing the game on L2 and making it more costly for both users. This issue can be mitigated by having a game reputation score which would stick with users between games and would be adjusted during a game finalization based on the amount of slashings that occurred in the game. Since we can't prove who of the 2 users was the malicious one both users would get slashed. This would not be that big of a deal because villains would naturally get weeded out of the game since they would receive the most slashings (and then they would lose all the frens and end up being sad and alone like peepo bellow).

[

image

1280×1280 89.9 KB

](https://europe1.discourse-cdn.com/business20/uploads/aztec/original/1X/a445e9f79f58fbc08249b082418e7e2b25ef017a.jpeg)

Solution to problem 2

Context

: "I am a user A and I realize that I did a bad turn n

rounds ago so I revert the game and choose a different turn."

Since user B would not accept the play at that point, the game would have to go on chain. Once malicious user A is forced on chain via the slash-revoke_slash scheme user B could call dispute(kernel_proof)

. The function would verify the kernel proof and check that the new state advanced further than the one submitted by user A.