(Thanks to Olivier Bégassat for his comments).

It is possible to increase the security of a rollup, by adding (1) an alert function that stops the rollup (or triggers a specific governance mode –that's outside the scope of this post) if someone can demonstrate that the circuit is not sound, and this whatever the current state of the rollup; (2) a time window to rollup's updates, so anyone can easily detect a suspicious state update and triggers the alert before the suspicious state can be used.

For context, a simplified zk-rollup contract is:

Rollup{ circuit, // the circuit of the proof, immutable currentRootHash // The current state of the rollup

```
// Update the state if the proof is valid
function update(
     oldRootHash, newRootHash,
     txData, // The list of the transactions
     proof // the proof that the transition is valid
    ) {

    if currentRootHash == oldRootHash and
        proof_is_valid(circuit, proof, oldRootHash, newRootHash, txData) {
        currentRootHash = newRootHash
    }
}
```

}

(1) Soundness alert function

This was briefly mentioned by VB in his presentation on multiprovers (applied here to a single prover rollup) It can also be considered as a kind of cryptographic canary, proposed by Justin Drake.

We add an extra safety function to alert in case of a soundness issue:

function soundness_alert(oldRootHash, newRootHash1, newRootHash2, data, proof1, proof2) { if newRootHash1 != newRootHash2 and proof_is_valid(circuit, proof1, oldRootHash, newRootHash1, txData) and proof_is_valid(circuit, proof2, oldRootHash, newRootHash2, txData) { // put rollup in emergency state } } }

This alert function does not depend on the current rollup's state: it just shows that we found a state and a set of data that lead to two different states.

Of course, there is no guarantee on data: it does not have to be valid transactions.

There is also no guarantee on oldRootHash: it can be the root hash of a state that is not reachable from the rollup's genesis state with only legal transactions. However, whatever this previous state and whatever the data, the circuit must ensure a deterministic outcome.

This function can be used by security researchers who found both a security issue and a way to exploit it. However, a black hat hacker will not use this function: he will directly use the rollup's update function to create a state which allows him to extract all rollup's funds. But we can secure the rollup against this attack by adding a time window.

(2) time window between updates

That is basically an optimistic rollup approach: the update is not valid immediately, giving time to anyone to interject, i.e. call the alert function. The difference with (1) is that instead of having to find an arbitrary soundness problem, the honest actors just follow the rollup and execute the transactions themselves. A different result with a valid proof means a soundness problem has been identified.

To illustrate, here is a possible (and naive –used for illustration only) way by changing the rollup state and update functions.

State{ circuit, windowTime, // minimum time between two updates currentRootHash, futureRootHash, // the root hash that can be challenged transitionProof, // proof of the transition currentRootHash -> futureRootHash updateTime // last time the rollup was updated }

And with currentTime() returning the current L1 time, the update function is changed to:

function update(oldRootHash, newRootHash, txData, proof) { if currentTime() > updateTime + windowTime and futureRootHash == oldRootHash and proof_is_valid(circuit, proof, oldRootHash, newRootHash, txData) { currentRootHash = futureRootHash futureRootHash = newRootHash transitionProof = proof updateTime = currentTime() } }

If a Byzantine actor updates the rollup with a state different from the state that would have resulted from a "legal" execution of the txData, this will be detected by all nodes executing the transactions locally. They call the soundness_alert described above, with L1 being the L1 state and localNewRootHash and localProof being the root hash and the proof calculated by the honest actor:

```
soundness_alert( L1.currentRootHash, L1.futureRootHash, localNewRootHash, L1.proof, localProof )
```

This comes with an important caveat: the signatures must be present in the txData, if not the other nodes can detect the problem, but cannot create an alternative proof –as the proof needs to include the signature verification.