By @bbrandtom and myself.

Introduction

- ZK-based scaling solutions may opt for off-chain data (Validium is a recently proposed name for these systems)
- Vitalik described an attack on such systems by withholding the data, and a proposed solution. We laterdescribed why it is unsatisfactory.
- We describe below a solution for this attack.

System Layout

- An off-chain operator batches tx and generates proofs attesting to the integrity of the entire batch.
- The system transitions from state s to s' iff it provides an on-chain verifier with a valid proof and a commitment to s'
- Data as mentioned above is stored off-chain
- Withdrawals are integral to the proof-based state transitions: the user sends a withdrawal request to the operator, who includes it in a future batch/proof
- The smart contract can freeze the system if it malfunctions:
- A user whose withdrawal request isn't serviced by the Operator raises a flag with the smart contract. The smart contract freezes the system if the Operator does not withdraw the requested funds (as manifest in a proof)
- Anyone who observes the unavailability of the off-chain data, can raise a flag with the smart-contract. This increases the likelihood of detecting such an attack in a timely fashion.
- A user whose withdrawal request isn't serviced by the Operator raises a flag with the smart contract. The smart contract freezes the system if the Operator does not withdraw the requested funds (as manifest in a proof)
- Anyone who observes the unavailability of the off-chain data, can raise a flag with the smart-contract. This increases the likelihood of detecting such an attack in a timely fashion.

Recovering from a system freeze

- Naive approach
- The system is reverted back to a recent state for which data is known, as recent as available. Any party can submit a request to become the new Operator. A request includes the id of the recovered state, and a bounty. An alternative Operator is nominated based on the list of proposals sorted by most recent state and sub-sorted by size of bounty.
- Problem: An attacker can withdraw funds, attack the system's data availability, and once the system reverts to a recent state, their funds (already withdrawn!) in the system are reinstated. Essentially a double-spend attack
- The system is reverted back to a recent state for which data is known, as recent as available. Any party can submit a request to become the new Operator. A request includes the id of the recovered state, and a bounty. An alternative Operator is nominated based on the list of proposals sorted by most recent state and sub-sorted by size of bounty.
- Problem: An attacker can withdraw funds, attack the system's data availability, and once the system reverts to a recent state, their funds (already withdrawn!) in the system are reinstated. Essentially a double-spend attack
- The MVR Solution
- The assumption is that we can recover to a state based on a history that took place not more than k batches ago.
- It is important to note that we recover not merely to a recent state of the system, state_recent, but rather to a state_new, which is a solvent state derived from state_recent
- In the normal course of affairs, each withdrawal is accompanied by the set of transactions that led to it in the last k state transitions. In the worst case, this would include all transactions in the last k batches. Assuming t tx/batch, that would mean k*t transactions would be included as public input to the proof that settles such a withdrawal request
- Computing state_new, to which the system recovers: state_new = state_recent + Σ (published transactions later than state_recent)
- This solves the double-spend attack as state_new reflects the withdrawal that took place sometime between state_recent and the present
- The assumption is that we can recover to a state based on a history that took place not more than k batches ago.

- It is important to note that we recover not merely to a recent state of the system, state_recent, but rather to a state_new, which is a solvent state derived from state_recent
- In the normal course of affairs, each withdrawal is accompanied by the set of transactions that led to it in the last k state transitions. In the worst case, this would include all transactions in the last k batches. Assuming t tx/batch, that would mean k*t transactions would be included as public input to the proof that settles such a withdrawal request
- Computing state_new, to which the system recovers: state_new = state_recent + Σ (published transactions later than state_recent)
- This solves the double-spend attack as state_new reflects the withdrawal that took place sometime between state_recent and the present

Observations

- Worst case, when activating the MVR protocol, we get the performance of a ZK-Rollup, where all tx data is published on-chain.
- With Fast Withdrawals, the marginal cost of operating a system with MVR is in fact 0. The operator can allow users to withdraw through the Fast Withdrawal mechanism, and replenish its on-chain Cookie Jar only from “aged” accounts, i.e. accounts which haven’t had any activity in the last k batches. For that reason, withdrawals from those accounts, need not be accompanied by any transaction history.

Open Matters

- We did not describe how to handle deposits made in the last k batches. This is left as an exercise for the reader.

Tom Brand & Avihu Levy