A commonly cited tradeoff between optimistic and zk-rollups is security versus programmability. Today, optimistic rollups like Arbitrum and Optimism support far greater programmability as they support a very large subset of the EVM (though they have not launched to mainnet yet). Yet, optimistic rollups may be vulnerable to game-theoretic attacks by L1 miners. Moreover, optimistic rollups rely on fraud proofs, which may also be vulnerable to censorship attacks by block validators. On the other hand, zk-rollups are immune to such attacks, but are much more difficult to write so-called “smart contracts” in zero-knowledge circuits. Loopring, for instance, wrote custom AMM circuits, and Hermez currently only supports ETH and token transfers. ZkSync reports that they have made great strides in porting the EVM to a zk-rollup but I’m not sure if they have released any details yet.

One suggested means to resolve this tradeoff is zk-MerkleWitnessAndSigRollup, a generic SNARK circuit for stateless contracts:

The difficulty of implementing complex snark circuits is thought to be a blocker for widespread adoption of zk-rollups until “generalized snarks” are practical (meaning one universal snark circuit that would, for instance, generate a proof of correct execution of any EVM code that’s passed as input). But if most of the throughput boost achieved by a zk-rollup is due to (i) and (ii) and not (iii), then it should be fairly straightforward for any dapp, even if it has contracts with a lot of complex logic, to adopt a generic zk-rollup circuit for state maintenance - call it the zk-MerkleWitnessAndSigRollup or the zk-PenultimateRollup - and get a big scalability gain today (or as soon as someone implements it ;).

It seems that the holy grail of generalised zk-rollups is EVM compatibility. Instead, I suggest a slightly different approach: a ZVM - that is - a VM whose opcodes can be executed by both

the EVM and a zk-rollup. Rather than attempt the herculean task of porting the EVM to a snark circuit, one could simplify the job and get the best of both worlds.

I have not fully fleshed out the benefits and drawbacks of this approach, let alone implementation specifics, but I would just like to put it out here for discussion. I am particularly curious if this approach could make flexible L1-L2 interoperability possible. For instance, if the rollup could access ZVM state on L1, or vice versa.

Thank you!