Thanks Aayush for feedback and discussions.

Zero-knowledge protocols often rely on a set of trusted participants to contribute to a common setup without leaking their secret. Let's take a look at two examples: trusted setups and a special case of nullifier generation.

Trusted setups

Proving schemes like Groth16 or KZG rely on toxic waste

, field elements that have to be randomly sampled then discarded for the scheme to stay secure. Instead of trusting a single party to generate and discard this secret, multiple participant can contribute using MPC. The trust assumption here is 1/N

: as long as at least one participant is honest and destroys their secret, the trusted setup cannot be broken. But if an attacker can get their hands on all of them, they can generate proofs for arbitrary inputs.

Once the trusted setup is done, there is no incentives for participants to discard their secret share other than reputation. Even if participants want to behave honestly, they often rely on a very limited number of tools for contribution that could be compromised. They are not incentivized to check the code or to use minority tools as even if their secret share is leaked, they incur no consequences.

One way to add cryptoeconomic guarantees to trusted setups would be to have participants restake, and publish a zk-SNARK proving that they contributed correctly to the setup and making the hash of their secret public. Anyone proving they know a preimage of the hash could slash the node and get some part of its stake as reward. The preimage does not have to be revealed publicly so as not to compromise the setup.

Nullifier generation in proof of identity applications

Another example is nullifier generation for proofs relying on attestations known by a centralized issuer. For many proof of identity applications like Proof of Passport, zk-email or anon-aadhaar, we want to produce public nullifiers deterministically derived from the attestations to avoid using the same attestation for multiple proofs. But doing that necessarily deanonymizes the user to the attester (e.g. government or mail server) if they kept a registry with all the signatures they have ever produced. See a more detailed exposition of the tradeoff space in this ethresearch post.

Instead of having a trusted server generating a nullifier as hash(attestation, salt)

, we can think of a network of nodes each having a share of a secret and computing together H(attestation, threshold secret)

. The threshold secret can be reconstructed from the secrets of t/N

nodes. The same can be done using threshold signatures instead of a hash function for simplicity. The requirements for the threshold secret are quite strong:

- It should continue being accessible as long as the circuit is used, otherwise new proofs can't be nullified.
- It should never be known by a single node.

This system is a bit more involved as it has a t/N

trust assumption rather then 1/N

and it involves nodes keeping their secret share safe instead of discarding it. Just like before, nodes can make a hash of their secret share public and prove they correctly contributed with it at each nullifier generation with a zk-SNARK. This guarantees the reconstructed secret is always the same, which makes the nullifier generation deterministic.

Slashing tradeoffs

In both cases, making secrets public is detectable but leaking secrets privately to a malicious party is not. An attacker could corrupt individual nodes until they get t/N

secret share and break the proving scheme or deanonimize users. But we can add economic security with restaking on EigenLayer. Specifically:

- It should be very costly for a node to leak its secret share.
- It should be very profitable for a corrupting node to reveal another node's secret and slash them.

Let's denote p

the portion of the slashed staked given as a bounty to the slasher.

- p

should be less than 1 or nodes could slash themselves, get their stake back and lose nothing.

- p

should not be increasing with the number of nodes slashed or an slasher would be incentivized to corrupt as many nodes as possible before slashing, which would bring the network closer to failure.

- p

should not be decreasing with the number of nodes slashed or a slasher would be incentivised to slash only a subset of nodes that have leaked their secret to them.

It thus seems to us that p

should be constant with the number of nodes slashed. The rest of the slashed stake can be burned.

EigenLayer could also be used here to slash nodes that repeatedly stop being live during a MPC operation. The overhead of MPC can be quite high. For synchronous operations like the nullifier generation scheme mentioned above, nodes aborting the operation could lead to significant delays.

This is all work in progress and there are a lot of open question:

- Are rotations possible in the nullifier generation scheme above so that new nodes can enter the network ?
- Is there a way to make sure that nodes leaving the network can’t leak their secret share at no cost ?
- What would be the optimal p

?

- Are there other such trusted operations that would benefit from cryptoeconomic security from restaking ?

We would love to hear more from the team and community about these questions and potential other uses of EigenLayer in zk.