This was asked on Telegram and previously on the forum, so it’s worth addressing how we’ve been thinking about this situation.

Generally-speaking, consensus protocols allow multiple nodes/actors in a network to agree on the result of a certain computation. However, this becomes tricky when the computation result is dependent on some randomness. This is known as non-deterministic execution (or randomized functionalities), and currently, most blockchains, Ethereum included, simply do not support these.

To further explain what randomized functionalities are, we can work with a simple example, such as the function x_plus_random(x)

, that takes an input x

and return x+r

, where r

is generated randomly at every invocation of the function. Now imagine we have a network of nodes. If we ask each node to independently run that function, each of them would likely receive a very different result, because r

changes from one call to another.

Given this, how can the network agree on a single outcome? Who’s to determine what the ‘right’ r

should be for a certain function call, and how do we communicate that to the rest of the network?

For the current release of our test-net, we randomly select a single worker to execute each computation. With the assumption that TEEs cannot be tampered with, we can support randomized executions natively, since there’s no need to reach consensus on an execution with other workers (each one works on a different problem, and Ethereum only validates the remote attestation proof).

However, working with a model of a single worker per computation has many disadvantages when it comes to availability, safety and performance. This is therefore one of the things we’re working to address in the next release of Discovery. Instead of a single worker, we plan to randomly sample a committee of workers for each secret contract and each epoch (epoch=several blocks long. Could be many), which is more in-line with how the original Enigma whitepaper worked.

Unfortunately, having each computation executed by a group instead of a single node gets us back to the deterministic nature of the consensus problem. To overcome this, here are a couple of proposed solutions:

1. Before the start of an epoch, share with each of the workers assigned to a committee a unique random seed. This seed could then be used in a PRG to generate as much randomness as the computations in the epoch require, in a way that doesn’t require any further communication between the workers.

Advantages:

a. Easy to implement. Likely the first approximation of a solution which will be used in Discovery.

b. Negligible performance overhead.

The main disadvantage is that if even one TEE is compromised throughout the epoch (which is hard, but not impossible), then the attacker can fully predict the randomness going forward in that epoch. This is potentially a concern for applications like lottery that have a lot of money at stake.

1. Implement a coin-tossing protocol - this is essentially an MPC protocol that allows multiple parties to agree on a shared randomness.

Advantages:

a. It practically solves (or at least greatly mitigates) the attack above, especially if the random value is revealed to the parties only at the time of the computation.

b. It’s cool

.

Disadvantages:

a. Longer development time. Won’t be ready in time for Discovery.

b. Performance - coin-tossing, like consensus, requires interaction between the workers which would slow down computations. The best way to mitigate this would be to pre-process the shared randomness, which would create other development challenges.

The working plan right now is to go with #1

for Discovery, and #2

for the next release when MPC is available. #2

will likely be broken down to several smaller milestones.