In designing decentralized systems, we often make tradeoffs between trust and other desirable features. It can be useful to think of this in a cost-benefit paradigm: making a trust assumption is a cost because it increases risk, but if we can get enough value in return it might be a good choice.

(All of this assumes that system designers are transparent with their communities, so that everyone is making informed decisions.)

Trust assumptions

A trust assumption is an assumption that some party, or some quorum of parties, is honest. For example, it might assume that Alice is honest, or that BigCorp is honest, or that at least seven of the ten members of some committee are honest.

Note that we're following the common convention of using "honest" as a shortcut. What we really require is that the parties are honest (aspiring to do the right thing), competent (able to do the right thing), and diligent (willing to put in the effort to do the right thing), so that we can trust that they will act appropriately as required by whatever protocol we're using.

A trust assumption is a statement like this: "If [some entity or set of entities] is dishonest, then [some bad outcome] can happen."

Trust efficiency

Trust efficiency is like cost efficiency, except that the "cost" reflects trust assumptions made in the design.

There are two main ways to make a design more trust efficient:

- get the same benefits while reducing trust assumptions, or
- get more benefits without increasing trust assumptions.

There's a close analogy to ordinary cost efficiency, where we're looking for ways to get the same benefits while reducing cost, or to get more benefits at the same cost.

Some design principles

Here are two basic principles of trust-efficient design.

Principle 1: Rather than making two separate trust assumptions, make the same trust assumption twice.

Suppose we have a system with two components, and a failure of either or both of the components would be equally harmful. If each component is going to make a trust assumption, it's better if both trust the same parties in the same way, instead of having separate trust assumptions.

As an example, suppose we're designing a rollup protocol that includes a data availability mechanism and a separate mechanism to validate execution. One approach would be to outsource data availability to a specialized service, accepting the trust assumptions of that service; and to separately recruit a permissioned set of validators for the execution mechanism, making the trust assumption that at least one of those validators is honest. That's two components with two separate trust assumptions.

A more trust-efficient approach would have the same parties who are providing data availability also provide validation services. That way, the data availability component and the validation component could be making the same trust assumptions. A failure of that one assumption would still be fatal—but that's better than having two separate assumptions with a failure of either one being fatal.

Principle 2: If you make a trust assumption at one point, take advantage of it elsewhere.

Suppose again that we have a system with two components, and one component makes a trust assumption whose violation would lead to some bad result. Having made this assumption, we should ask whether there are additional benefits we can get from making the same assumption in the other component.

Let's consider an optimistic rollup protocol with an external data availability mechanism that make some trust assumption. In this example the rollup will have permissionless validation, so its execution level will only assume that some honest party is willing to participate in validation.

The design has already adopted the trust assumption of the data availability component, and accepted that a violation of that assumption would be fatal to the chain. Having accepted that risk already, we should look for opportunities to take advantage of that trust assumption in the execution/validation part of the design. Put another way, when we design the execution/validation sub-protocol, we can make the same trust assumption "for free".

And there is a great opportunity to take advantage of that assumption, by allowing a quorum of the data availability quorum

to do instant confirmation of execution claims, bypassing the usual one-week delay for confirmation. This could be misused by a quorum of the data availability committee to cause fatal harm to the chain, but that doesn't make things any worse because that quorum could already cause fatal harm by abusing its data availability powers, if it turned malicious.