Appendices

A MAKING CREDIBLE THREATS - ALTERNATIVE APPROACH

The other way for the Customer's to make credible threat that it will check n inputs is described here. First, the Customer must process n of the λ_Δ inputs generated during an allocation and send the outputs to the Verifier. The Verifier then compares those outputs with outputs provided by the Supplier. However, since the Supplier sends its output on the output channel, which the Customer can read [10], the Customer can simply copy it and send it to the Verifier, making the Customer's threat of processing the inputs non-credible. To make the threat credible the Customer must commit to the output before the Supplier's output is available. To accomplish this for each input the Customer randomly [11] chooses to process it, hashes the output and sends it to the Supplier. To prevent the Supplier from using this information to only process when the Customer sends a hash, the Customer sends hashed random data for each input. The protocol then is: for each input the Customer generates a hashed output and sends it to the Supplier who then sends its output. Then, as before, at the end of the allocation the Customer provides a list of which outputs it processed to the Supplier by sending $\alpha(O_c)$. Then to determine which inputs are the test inputs the Supplier hashes all of its outputs and find which hashes are common to the two sets. We call that subset of Supplier outputs O_v where $O_v = \{o_i : hash(o_i) \in \alpha(O_c) \cap \alpha(O_s)\}$. This is because if the Customer sent $\{o_i : o_i \in O_c\}$ then the Supplier could skip processing the inputs and simply use the outputs provided to produce O_v. Or if the Customer sent the index i of each test output the Supplier could neglect processing the inputs until it received the indices to produce O_v.

At the end of the allocation the Customer also sends O_c to the blockchain as $\Gamma(O_c)$. The output set is double hashed because if the Customer sent $\gamma(O_c)$ then the Verifier on the blockchain would have to hash all of the Supplier's n outputs which is expensive. Instead the Customer sends $\Gamma(O_c)$, requiring the Supplier to send $\gamma(O_v)$ to the Verifier. Then the Verifier only has to hash a single element to compare against the Customer's output hash: $\Gamma(O_c) == hash(\gamma(O_v))$. The Verifier then compares the two hashes, if they do not match then the Verifier requests Mediation, otherwise it distributes payment. The drawbacks of this approach are: 1) the Customer's trusted Supplier must be online during the entire allocation to send the outputs, 2) the output from the Supplier would be delayed by at least(most) (time for trusted Supplier to construct an output + time to transfer output to Supplier −(+) time required for Supplier to process input).

[10] the Customer must be able to read these outputs, otherwise we lose the benefits of streaming
[11] such that the Customer processes n inputs total