3.2 Simplifying Complex Components

Assume that we have a component which has a considerable arithmetic circuit complexity. If this complex component is needed in some arbitrary computation steps, we will need to repeat its circuit in every step of the computation. This will increase the complexity of proof generation considerably. Fortunately, there is a workaround. Instead of repeating the circuit of the component, we can use a simpler cryptographic hash calculator circuit during the computation, and use a final verification circuit at the end to verify the functionality of the component.

Every component accepts an input *in* and produces an output *out*. As a result we can denote the component by a deterministic function that maps an input sequence with any arbitrary length k, to an output sequence with the same length:

$$f((in_1, in_2, \ldots, in_k)) \coloneqq (out_1, out_2, \ldots, out_k)$$

In every step of the computation, we replace this component with a cryptographic hash calculator which receives *in* and *out* as its inputs and gets activated in the same computation steps that the component must be active. When the hash calculator is active, it hashes its inputs, so at the end of the computation it has computed a digest:

$$h(in_1, out_1, in_2, out_2, \ldots, in_k, out_k) \coloneqq digest_f$$

where k is the number of steps that our component must have been active.

Now the prover needs to prove that he knows values $in_1, out_1, \ldots, in_k, out_k$ such that they are correctly produced by the component:

$$f((in_1, \ldots, in_k)) = (out_1, \ldots, out_k)$$

and their digest is also correct:

$$h(in_1, out_1, \ldots, in_k, out_k) = digest_f$$

where functions f and h, are both known to the prover and verifier.

The circuit for verifying this assessment usually is straight forward. When the computation involves a large number of steps and the component is complex, this approach can reduce the cost of proof generation considerably. Memory components are good candidates for being simplified by this method.

Interestingly, this approach can also reduce the number of computation steps. When we use a hash calculator circuit instead of our component, *out* will be available in the same computation step that *in* is available. This will eliminate the computation step that is needed for generating *out* by the component's circuit.