Sakura: A Flexible Coding for Tree Hashing

Guido Bertoni\textsuperscript{1}, Joan Daemen\textsuperscript{1}, Michaël Peeters\textsuperscript{2}, and Gilles Van Assche\textsuperscript{1}

\textsuperscript{1} STMicroelectronics
\textsuperscript{2} NXP Semiconductors

\textbf{Abstract.} We propose a flexible, fairly general, coding for tree hash modes. The coding does not define a tree hash mode, but instead specifies a way to format the message blocks and chaining values into inputs to the underlying function for any topology, including sequential hashing. The main benefit is to avoid input clashes between different tree growing strategies, even before the hashing modes are defined, and to make the SHA-3 standard tree-hashing ready.

\textbf{Keywords:} hash function, tree hashing, indifferentiability, SHA-3.

\section{Introduction}

A \textit{hashing mode} can be seen as a recipe for computing digests over messages by means of a number of calls to an underlying function. This underlying function may be a fixed-input-length compression function, a permutation or even a hash function in its own right. We use the term \textit{inner function} and symbol $f$ for the underlying function and the term \textit{outer hash function} and symbol $F$ for the function obtained by applying the hashing mode to the inner function.

The hashing mode splits the message into substrings that are assembled into inputs for the inner function, possibly combined with one or more chaining values and so-called frame bits. Such an input to $f$ is called a \textit{node} \cite{6}. The chaining values are the results of calls to $f$ for other nodes.

Hashing modes serve two main purposes. The first is to build a variable-input-length hash function from a fixed-input-length inner function and the second is to build a tree hash function. In tree hashing, several parts of the message may be processed simultaneously and parallel architectures can be used more efficiently when hashing a single message than in sequential hashing \cite{17,22,3,9,6}.

The motivation for standardizing a tree hash mode, or to have a tree-hash-ready SHA-3 standard, was discussed at various occasions during the SHA-3 competition on the NIST hash-forum mailing list \cite{18}. A few candidates, like MD6, SANDstorm and Skein, proposed built-in tree hash modes \cite{21,23,10}. At the Third SHA-3 Candidate Conference, Lucks, McGrew and Whiting motivated why the SHA-3 standard should support parallelized tree hashing \cite{15}.

Different applications or use cases call for different approaches to tree hashing and different tree topologies. For instance, some environments favor cutting the input message in consecutive pieces and hashing these pieces independently, while others favor to hash interleaved pieces of data, see, e.g., \cite{11}. In his presentation at ESC 2013, Lucks suggested to use a $n$-ary tree with much potential...
parallelism and to let the implementation choose the most appropriate evaluation strategy [14]. As another example, some applications require to keep the intermediate hash values (e.g., to be able to re-compute the digest if only a part of the input changes), whereas the mere exploitation of parallelism does not require it.

Given all this diversity, it seems difficult to agree on a “one-size-fits-all” tree hash mode. Instead, we take the different approach of allowing different tree hash modes to co-exist. However, the co-existence of different modes on top of existing (serial) hash functions calls for caution. While each individual hash mode can be proven secure, the joint use of several modes can become insecure, in particular due to the different coding conventions that could collide into equal inputs to the inner function. This paper proposes a way to bring together different tree hash modes in a secure way and follows ideas presented in [5, Slides 54-59].

We show that it is possible to define a tree hash coding, i.e., a way to format the input to the inner function, that can cover a wide range of tree hash modes. For a carefully designed tree hash coding, one can prove that the union of all tree hash modes compatible with it is sound. By sound we mean that it does not introduce any weaknesses on top of the risk of collisions in the inner function. More precisely, a hashing mode is sound if the advantage of differentiating \( F \) from a random oracle, assuming \( f \) has been randomly selected, is upper bound by \( q^2/2^{n+1} \), with \( q \) the number of queries to \( f \) and \( n \) the length of the chaining values [1,16,7,6].

As a result, tree hash modes compatible with the defined coding can be progressively introduced while preserving their joint security. Also, as an additional benefit, a tree hash mode following the coding convention is sound by construction, without the need of additional proofs.

For proving soundness, we use the results of [6], in which we specify a set of conditions for a tree (or sequential) hashing mode to be sound. We assume that to the choice of \( f \) is attached a security parameter, like the capacity in the specific case of sponge functions or the security strength [19,2]. We consider this security parameter to be specified together with \( f \) and to remain constant for its entire use in a tree hash mode.

The remainder of this paper is structured as follows. In Section 2 we explain the range of possibilities of our proposed sound tree hash coding and illustrate it with some examples. In Section 3 we specify Sakura, the coding we propose, while in Section 4 we define what it means for a hashing mode to be compatible with Sakura and prove that any such tree hash mode is sound. In Section 5 we give some examples of modes and in Section 6 we provide a concrete proposal in the context of making the SHA-3 standard tree-hashing ready.

## 2 Functionality Supported by Sakura

We start by recalling the very general concept of node and tree of nodes. We then capture the functionality of Sakura with trees of hops and how nodes and hops relate to one another. Finally, some figures illustrate the concepts.