Tree Hashing

a simple generic tree hashing mode designed for SHA-2 and SHA-3, applicable to other hash functions

Stefan Lucks

Bauhaus-Universität Weimar

January 11, 2013

What this talk is about . . . and what it isn't!

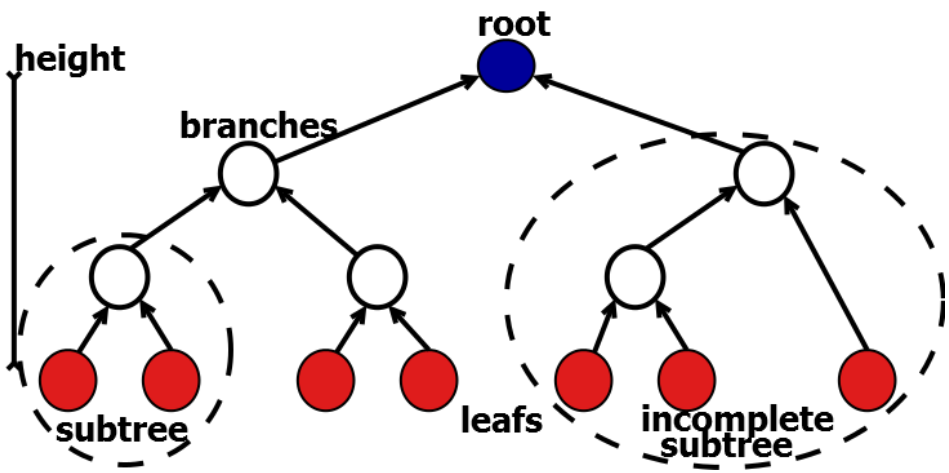
- this **is not** about a paper already written
- this **is not really** about new ideas or results on tree hashing
- this **is** a re-hash of known results and ideas
- this **is** about one standard tree hashing mode
  - for both SHA-2's
  - as well as for SHA-3
- I'll discuss
  - alternative solutions and their disadvantages
  - different primitives (compression fn. versus full hash)
  - different tradeoffs on parameter choices
  - . . .
- I am interested in your opinion on these issues . . .
- . . . and **I wouldn't mind to find co-authors for some proposal**

Tree Hashing – an Overview

Tree Hashing
- Introduction
- Alternative Solutiuons
- A Possible Tree Hashing Mode
- Discussion
- (Security Analysis)

Introduction: Tree Hashing Deals with Hash Functions

whose **data flow** from the **leafs** to the **root** of a graph-theoretical tree:



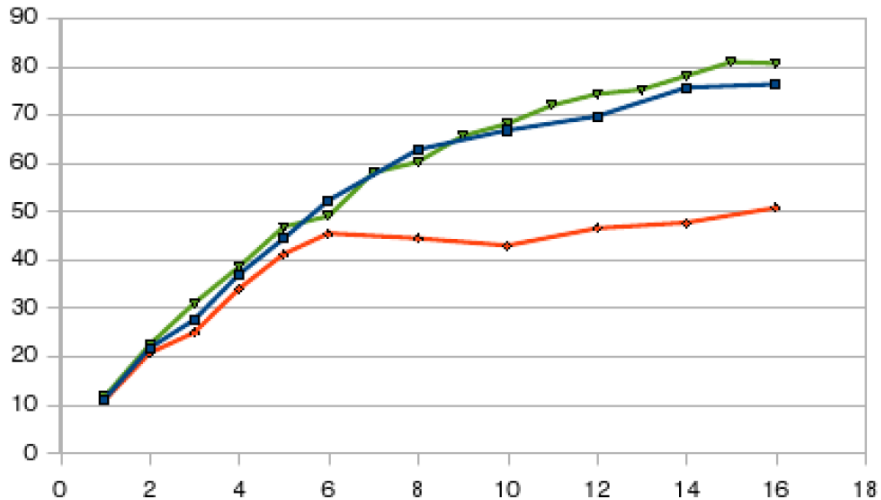
- has already been proposed by Merkle and Damgård (1989)
- has been an optional or integral part of several SHA-3 candidates (MD6, SANDstorm, Skein, . . .)
- with some theoretical analysis (MD6, Skein)
- has also been theoretically studied by the Keccak team

Motivation

does the world really need a standard for tree hashing?

1. parallelism (multi-core, distributed, "cloud")
2. fast hash recomputation, after small message changes
3. verify hash without reading all message blocks (Merkle/Lamport signatures, timestamping, . . .)

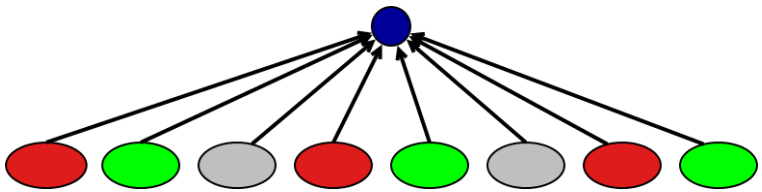
Performance results for MD6 tree hashing on 1–16 cores.
Red line: Small file.



Alternative Solutions: Clustering and Interleaving

- discussed on the SHA-3 mailing list (Shay Gueron, Dan Bernstein)
- internally discussed by the Skein team, during the design phase:
  1. full tree hashing seems complicated
  2. ideas for simplified tree hashing
     - Clustering (like Dan)
     - Interleaving (like Shay)

Clustering



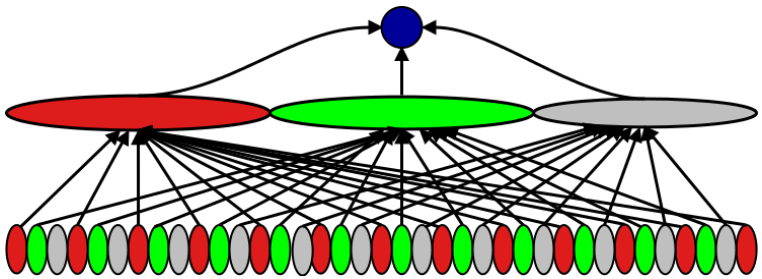
- group message into size-s clusters
- hash each cluster individually
- concatenate and hash results

- price for sequential implementation: double memory *(this is cheap!)*
- linear speed-up for huge messages
  - if clusters are large enough
  - and there are many clusters

where "large" and "many" grow with the number of machines

good cluster size s depends on (# cores)

Interleaving



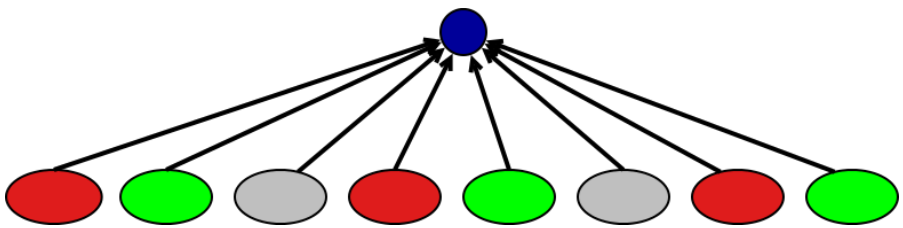
- split message into small blocks
- on each of t machines: hash every t-th block
- concatenate and hash the results

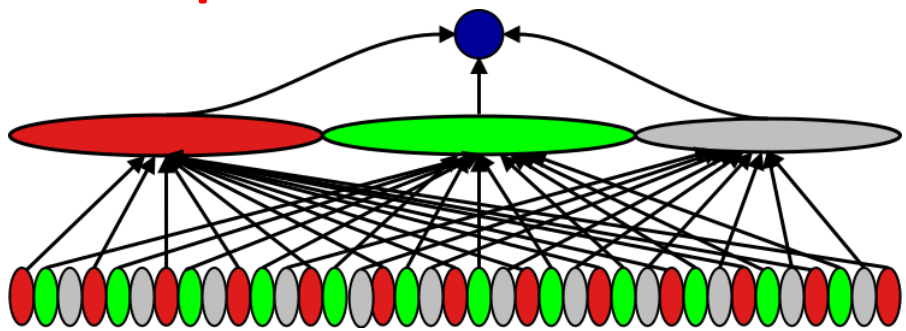
- friendly to SIMD implementations
- linear speed-up, even for medium-sized messages
- price for sequential implementation: t-times memory **(not cheap!)**

What is the problem?

no good candidate for a single standard

different topologies = mutually incompatible hash functions

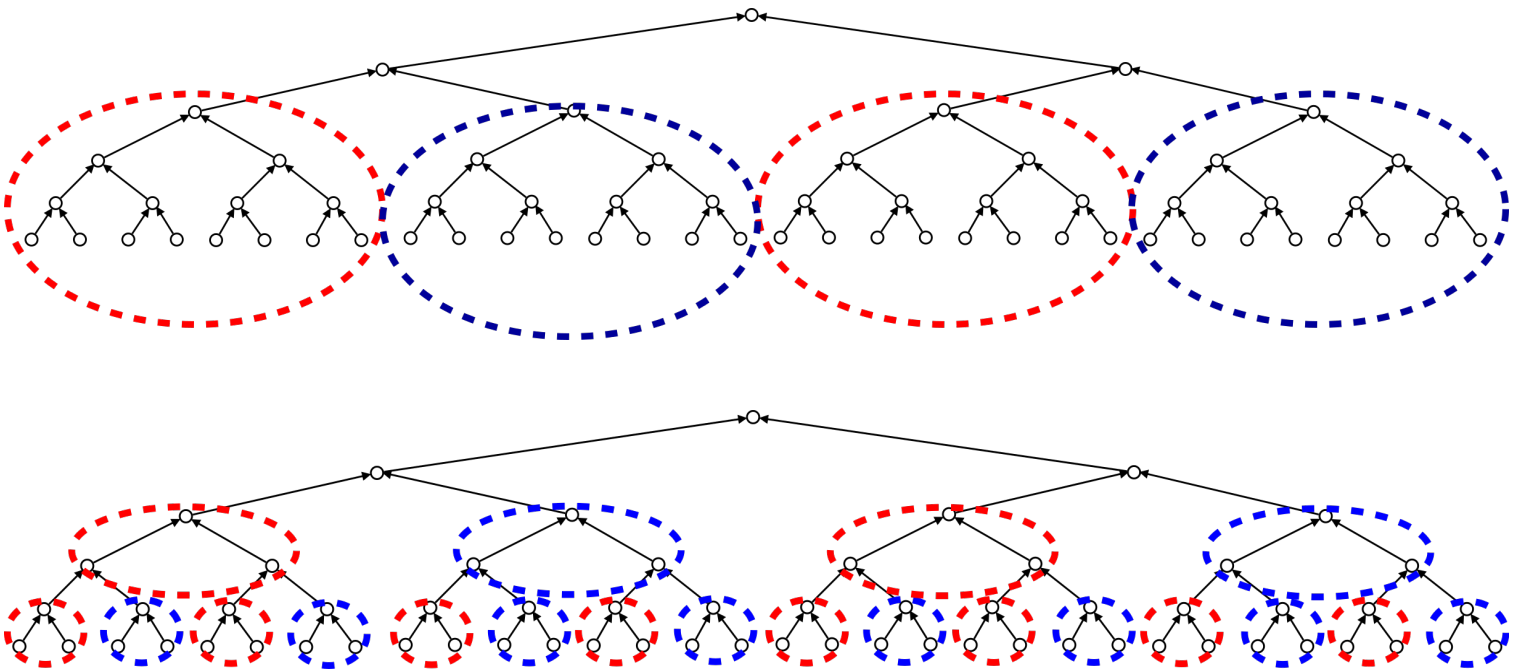




- clustering and interleaving are fundamentally different
- change of ruling parameter (s or t) = change of tree topologie

More Flexible: "Normal" Tree Hashing

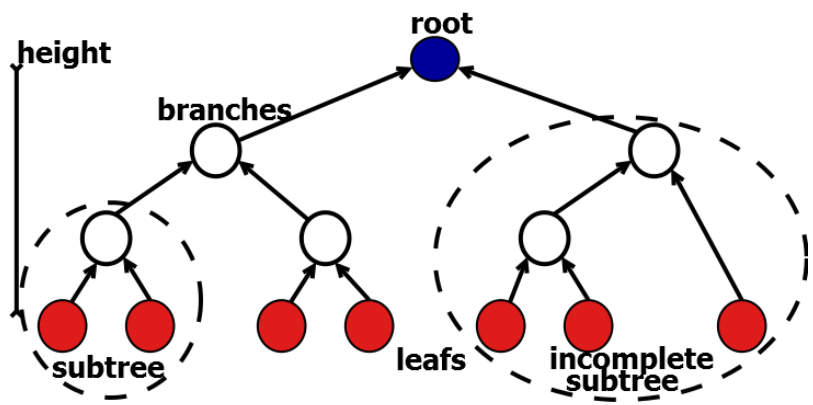
one tree topology, free choice for evaluation strategy, not sensitive to (# cores)



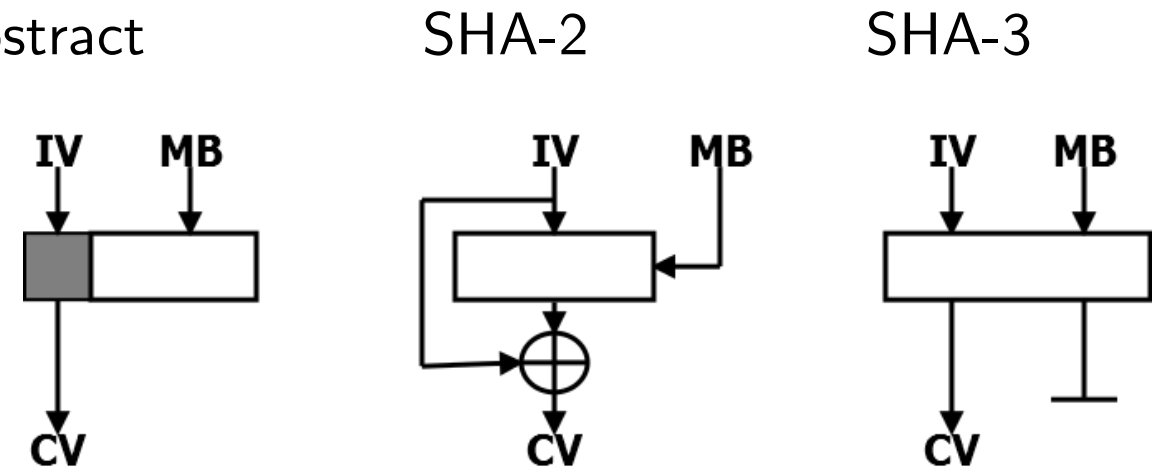
A Possible Tree Hashing Mode

as simple as possible, but not simpler (Albert Einstein, supposedly)

- use internal compression function
  (alternatively: the full hash function, discussed later)
- powers of two rule
  - split message into fixed-size chunks of $2^{\text{something}}$ bit (except for the final chunk).
  - All (complete) subtrees deal with 2^{whatever} bit.
- *domain separation* between
  - **leafs**, taking **MB**s as the input,
  - **branches**, taking **CV**s from leafs or other branches as the input, and
  - the **root**, being responsible for the final output transform.



The Internal Compression Function

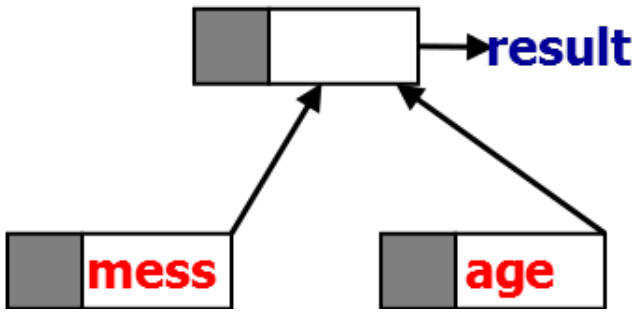


in:	m-bit message block (**MB**)	$m \in \{512, 1024\}$,	$m \in \{512, 1024\}$,
	n-bit initial value (**IV**)	$n = m/2$,	$n = m/2$ possible
out:	n-bit chaining value (**CV**)	not invertible	invertible

Sequential vs. Tree Hashing

sequential hashing:

tree hashing:

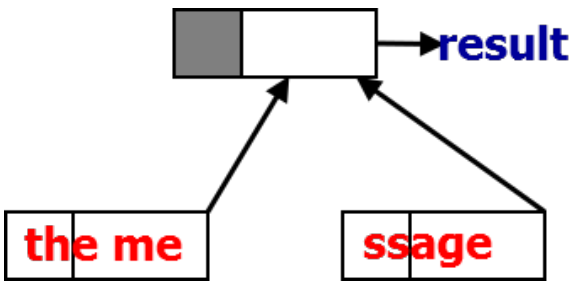


- # sequential compr. fn. calls = # leafs

Processing branches and root is overhead!

Avoiding the Overhead

use **IV**-field for larger **MB**:

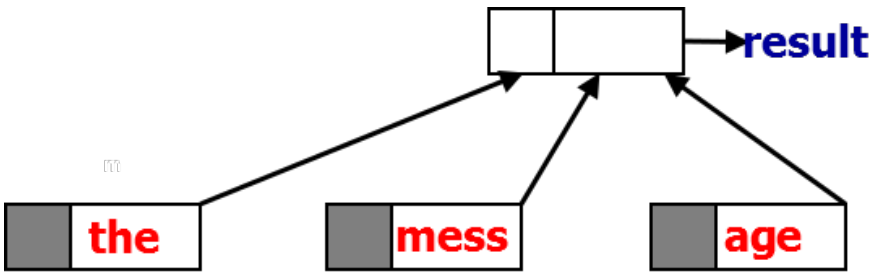


- SHA-2: OK, in principle

  but weaker than sequential construction

  (pseudocollisions $\longrightarrow$ collisions)
- SHA-3: insecure

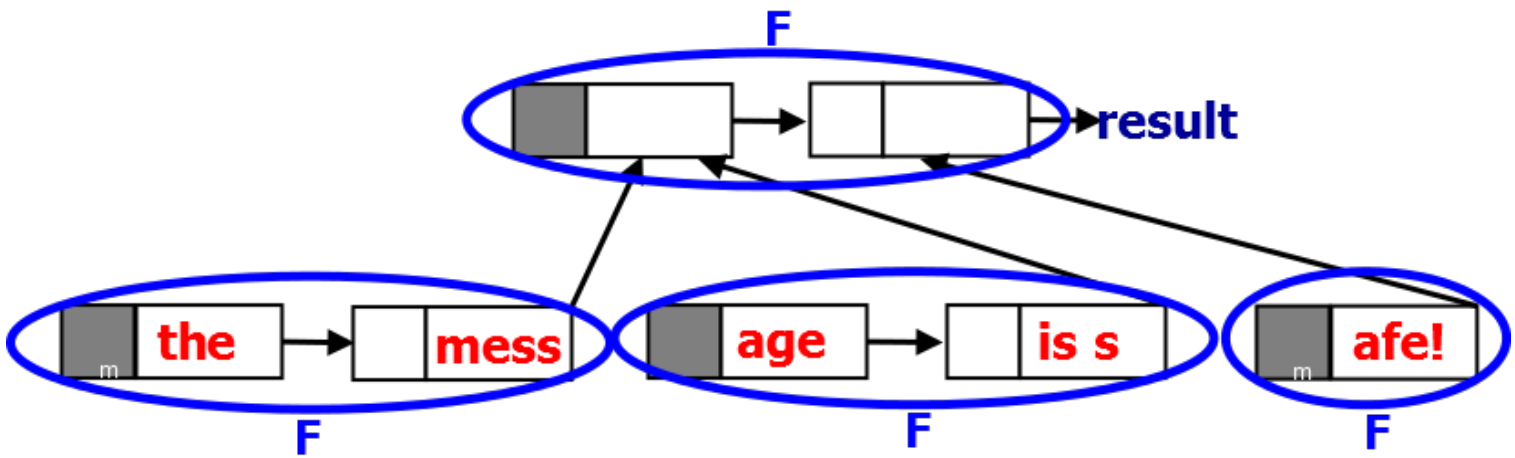
use **IV**-field for additional **CV**:



- security seems to be OK
- but "odd" subtree sizes (for SHA-2 and -3, that is)

Actually Reducing the Overhead

- "bigger" leafs and branches by iterating the compression function
- tantamount to going from binary to higher order trees
- transition from binary to 4-ary avoids more than half of the overhead
- gain from 4-ary to, say, 8-ary or 16-ary is smaller



- note the "inner hash function", **F**:
  - inputs of different lengths (e.g., "**the mess**" and "**afe!**"),
  - though lengths are a multiple of m (here: four characters)
  - Merkle-Damgård, but no MD-strengthening **(!!!)**
  - we can prove the soundness of tree hashes using **F**, assuming the compression function C is secure

Zero-Padding, Arity λ, Three Initial Values

- **zero-padding** $M_i := \mathrm{ZP}(M)$
  append $j < n$ zero-bits, such that m divides the length $|M_i|$ of M_i.

- **arity** $\lambda = 2^i$ (with $i \geq 1$)
  write $M_i = (M_{i,1}, M_{i,2}, \ldots, M_{i,k_i})$ as a sequence of $k_i - 1$ $(2\lambda m)$-bit blocks, followed by one block of length $\ell m \leq 2\lambda m$

- **main initial value** $\mathrm{MAIN} \in \{0,1\}^n$
- **derived initial values**

$$\begin{array}{lcl} \textbf{LEAF} & := & C(\mathrm{MAIN}, \text{“leaf”}). \\ \textbf{BRANCH} & := & C(\mathrm{MAIN}, \text{“branch”}). \\ \textbf{ROOT} & := & C(\mathrm{MAIN}, \text{“root”}). \end{array}$$

Tree-Hashing a Message M

$M_0 := \mathrm{ZP}(M)$

$M_1 := \mathrm{ZP}\Big(F(\mathbf{LEAF}(M_{0,1})) \,||||\, F(\mathbf{LEAF}(M_{0,k_0}))\Big)$

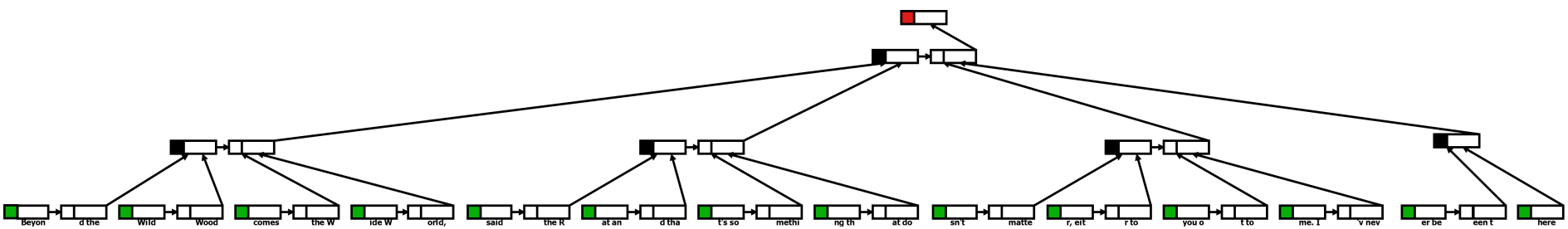
$i := 1$

while $k_i > 1$:

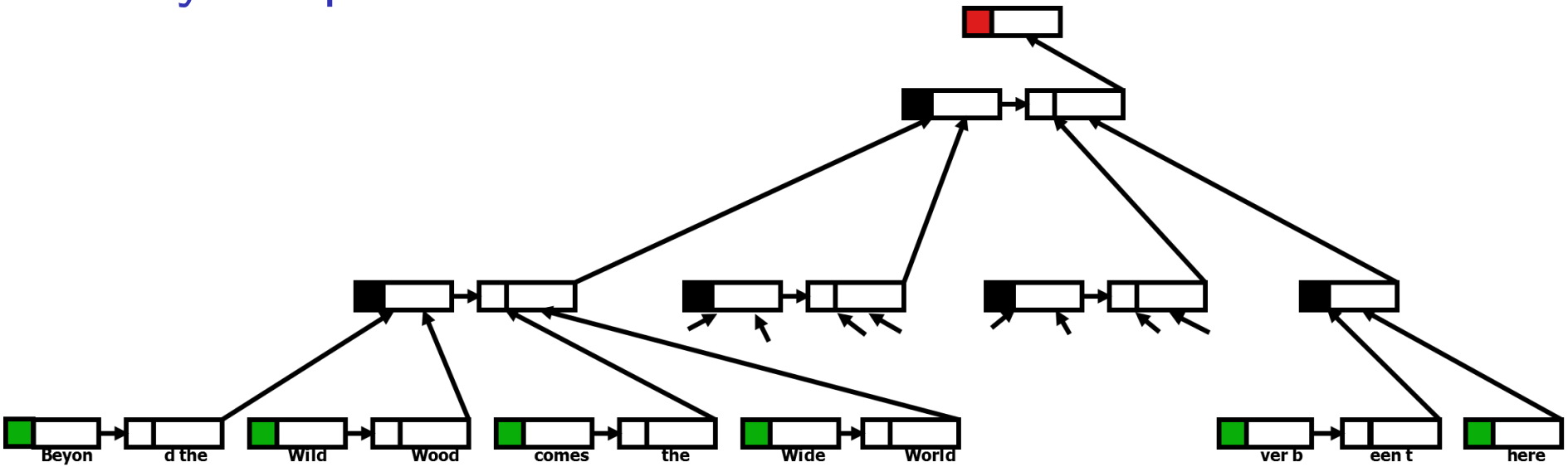
$M_{i+1} := \mathrm{ZP}\Big(F(\mathbf{BRANCH}(M_{i,1})) \,||\cdots||\, F(\mathbf{BRANCH}(M_{i,k_0}))\Big)$

$i := i + 1$

return $C(\mathbf{ROOT}, (\text{Parameters} \,||\, |M| \,||\, M_i))$



Security Properties



- If the compression fn. C is collision resistant, then so is our mode.
- If the compression fn. C is preimage resistant, then so is our mode.
- (Proving a similar claim for 2nd preimage resistance may be tricky.)

- Based on theoretical analysis from the Keccak team, one can prove this mode to be sound (indifferentiable from a random oracle). The final transform (using textcolorred**ROOT**) prevents length extension.

Discussion: 1. Hash Versus Compression Function

Points against using the compression function:

- a bit more complicated than using the full hash
- implementing tree hashing on some legacy systems may be difficult
- confusing for non-experts: the "compression function" is not explicitely defined in the (SHA-2) standard

Points in favour:

- more efficient (full hash $\longrightarrow$ padding $\longrightarrow$ more compr. fn. calls)
- if we use a tree-hash-specific MAIN initial value (to avoid trivial collisions between sequential and tree hashing), plain access to the sequential hash function would not work, anyway

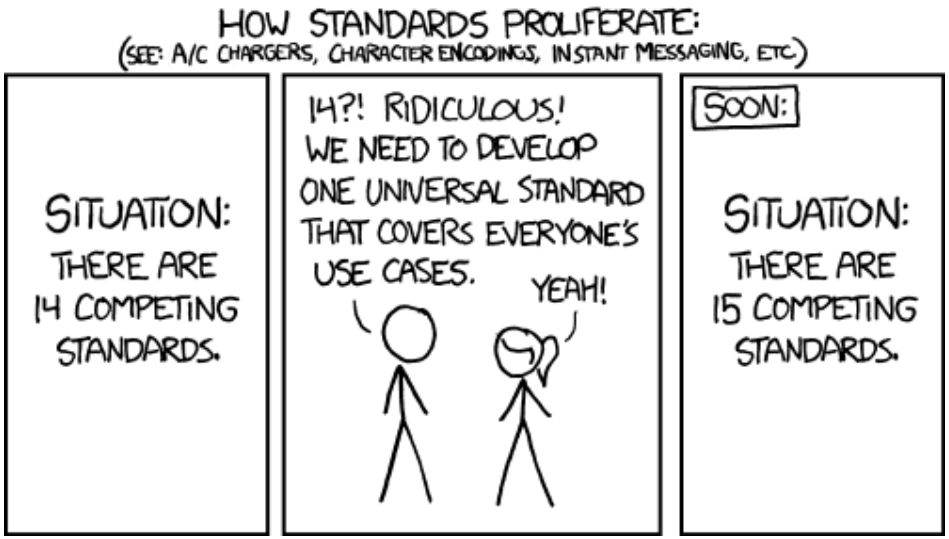
Discussion: 2. Parameters

The Skein hash mode supports three parameters:

- a **leaf arity** (λ for M_0),
- a **branch arity** (λ for M_i, $i > 0$), and
- a **maximum depth** d, such that M_d is hashed sequentially.

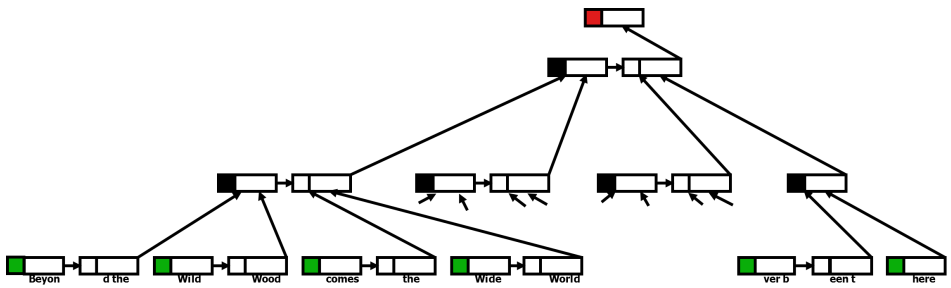
MD6 also allows to choose **maximum depth** SANDstorm fixes it at 4.

How many of these parameters would a good standard really need?



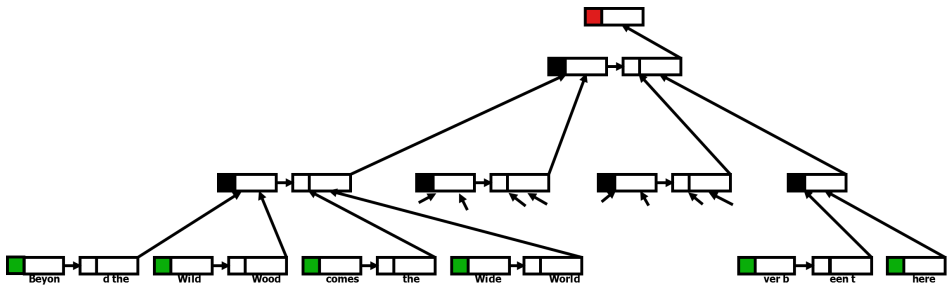
http://xkcd.com/927/

Leaf Arity and Branch Arity



- do we really need a different λ for leafs and brachnes?

Maximum Depth



- seems to make sense to save memory-constrained implementation from running out of memory
- but is hashing huge messages an issue for memory-constrained implementations?

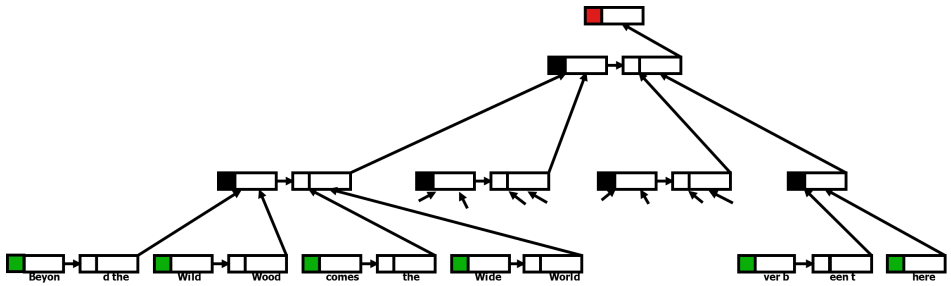
$$\text{memory} \approx \log_{\lambda}(\text{message length})$$

Which λ?

changing λ = changing tree topology = incompatible hash fns

- small λ:
  - \+ flexibility: much support for different application needs
  - − overhead: lots of compression fn. calls
- large λ:
  - − less flexibility
  - \+ less overhead
- What is the right tradeoff for a good standard?
- Or do we need to support (a restricted number of) different choices for λ?

Discussion: 3. Other issues



- should tree hashing include support signature- and timestamping applications (perhaps a variant with $\lambda = 2$)?
- how about support for variable output sizes?
- other features/properties you are missing?

Your Comments will be Greatly Appreciated!

Security Analysis

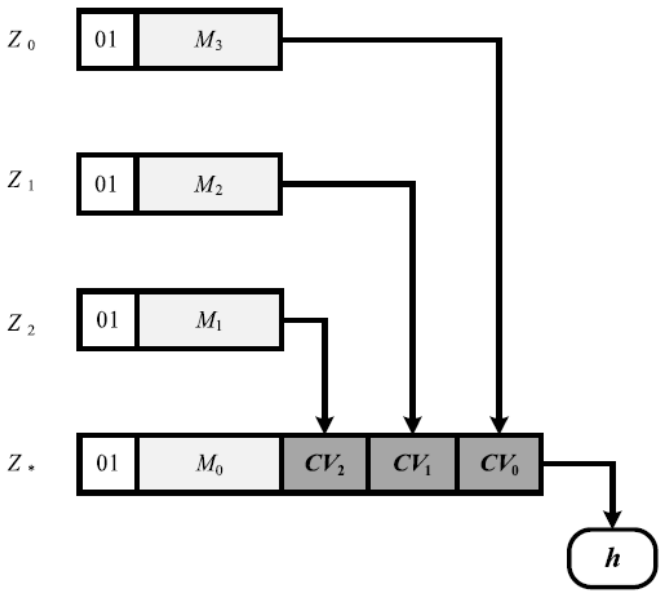
Bertoni et al, 4 sufficient conditions for sound tree hashing (eprint 2009)

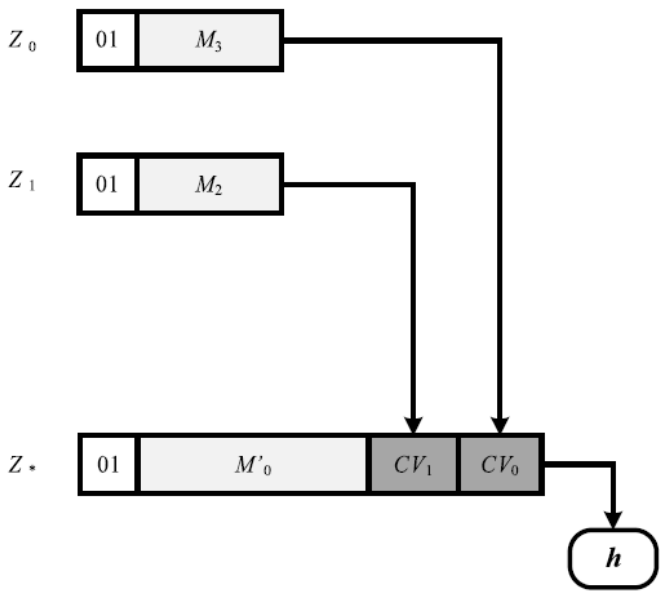
0. The tree topology (or "tree template") is defined by some parameters (in our case λ) and the length $|M|$ of the message.
   It does not depend on the actual content of M.
1. T is tree-decodable. ($\rightarrow$ next slide.)
2. T is message-complete. (Assume M has been (tree-)hashed. Given a transcript of the all calls to C, one can uniquely determine the message M.)
3. T is parameter-complete. (Given the same transcript, one can uniquely determine the parameters.)
4. T enforces domain separation between the root and the other nodes.

Up to the birthday bound, our proposed mode satisfies all these criteria, and thus is sound (i.e., indifferentiable from a random oracle).

Tree Decodability

The formal definition is quite complex. But the intention is, that, given any call $C(X, Y)$, the adversary cannot actually change turn values in Y are either MB or CV or meta-information, and the adversary cannot change this without actually changing X. Example:

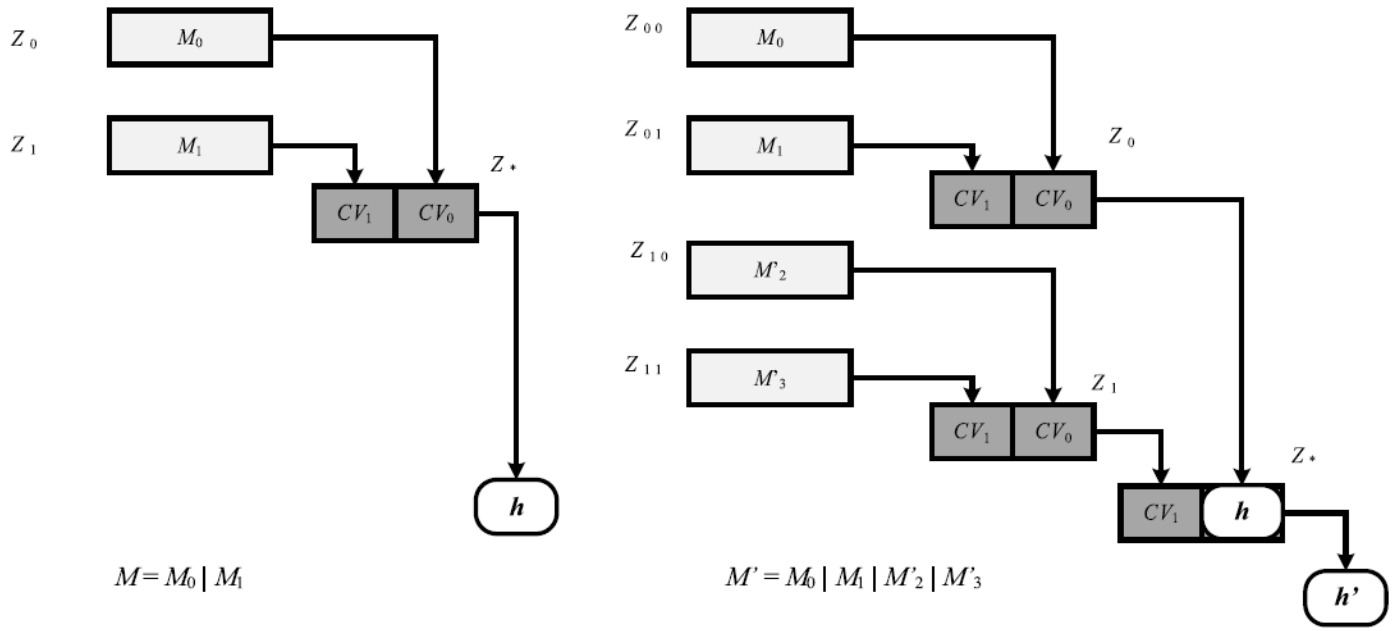




Our usage of **LEAF**, **BRANCH**, and **ROOT** prevents such attacks.

The Need for Domain Separation Between Root and Rest

without a "finalization" step, some generalized length extension is possible



We use **ROOT** only as the IV for the final transform.

Classical Security

- If C is **preimage resistant**, then so is our mode.
- If C is **collision resistant**, then so is our mode.
- Preserving 2nd preimage resistance may be difficult – in spite of claims by Bertoni et al.