Dear all,

recently, new Grøstl implementations have been developed and added for benchmarking in eBash. In addition to the table-lookup based approach, there are at least 3 other (cache-timing resistant) methods to implement Grøstl:

* cache-timing resistant Grøstl implementation using the Intel AES-NI instructions running at 13.8 c/b (~1.5x faster than table-based [1])

* cache-timing resistant bitsliced Grøstl implementation *without* the need of parallel message blocks (only ~1.5x slower than table-based on Core 2 Duo [2])

* cache-timing resistant vperm based Grøstl implementation (only ~1.2x slower than table-based on Core i5 [3])

Furthermore, the Grøstl implementation of the portable C library sphlib is now running correctly in eBash. The sphlib submitted to eBash did not compile on all machines until a month ago. Thanks to Thomas Pornin for fixing this bug.

This gives improved performance results on many machines benchmarked recently (e.g. [1], supercop version 20101111 or newer), and renders some comparisons made more than a month ago invalid. Additionally, we are currently working on improved Grøstl implementations for 32-bit platforms.

Also, an efficient 8-bit implementation for ATmega163 running at about 450 c/b is available from the Grøstl website [4]. Furthermore, Grøstl-256 and Grøstl-512 can be combined with small overhead in hardware since the main transformations SubBytes and MixBytes are exactly the same.

The new results confirm the balanced implementation characteristics of Grøstl throughout the whole spectrum from 8 to 128 bits, and show some further optimization potential.

Note that also tweaked constants (as announced at the Second SHA-3 Candidate Conference) will affect these performance numbers only in a negligible way.

Kind regards,
the Grøstl team

[2] written by Stefan Tillich and benchmarked in eBash
Martin,

Can you point out a reference to the new constants?

--John

Martin Schläffer

Martin,

Can you point out a reference to the new constants?

--John

Martin Schläffer

Martin,

Can you point out a reference to the new constants?

--John
John,

we are currently preparing a document describing the tweak in full. For the time being, we offer the following explanation.

One of the main design criteria of Grøstl was its simple design strategy to facilitate cryptanalysis and encourage external analysis. Indeed, this attracted lots of attention and led to the invention of rebound attacks [1], its extensions [2,3,4] and to the internal differential attacks [5,6].

None of these results pose any threat to our security claims of both the compression function and hash function of Grøstl. However, by simply changing the constants we can increase the security margin with negligible performance penalties.

If Grøstl gets to the next round we will:

* significantly increase the size of the round constants to make the internal differential attack and its extensions impossible
* and use different rotation constants in Q to make P and Q more different which further increases the security margin by one round.

All previous cryptanalytic work on Grøstl can still be used and applied immediately to the tweaked variant. Using Grøstl-256 (10 rounds) as an example, we get the following improved security margins:

* the best attack on the reduced-round Grøstl-256 hash function decreases from 6 rounds to 3 rounds
* the best collision attack on the reduced-round Grøstl-256 compression function reduces from 7 rounds to 6 rounds.
* the approach of [6] to construct non-random properties of the compression function and permutation is thwarted (even though they are irrelevant for the security of the Grøstl hash function)

To summarize, the performance and storage impact will be small, while we increase the security margin.

Kind regards,
the Grøstl team