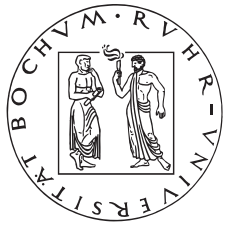
Evaluation of SHA-3 Candidates for 8-bit Embedded Processors

SHA-3

Ingo von Maurich, Alexander Wild, Cornel Reuber, Johannes Rave, Thomas Poeppelmann, Stefan Heyse

Horst Goertz Institute for IT-Security
Ruhr-University Bochum

The Second SHA-3 Candidate Conference



Implemented Hash functions

- BLAKE
- BlueMidnightWish(BMW)
- Grøstl
- Lane did not pass to round two
- SHAvite-3

All hash functions were implemented with 256 bit digest size. Pure AVR ASM code with user friendly C-Wrapper.

Problem: How to compare embedded implementations?

- C vs. ASM vs Java
- Speed vs. space optimizations or any trade-off in between
- ABI: Is message hand over at once or in small chunks?
- Memory alignment in single hash function or linked in library.
- Scaling factors for RAM, Flash, throughput
- ...

These implementations are neither optimized for speed nor for size.
Ram usage includes 64 byte message buffer.

Target

- ATmega163 CPU
- 130 Powerful Instructions
- 32x8 General Purpose Registers
- Up to 8 MIPS at 8 MHz
- 16 KByte Flash Memory
- 1024 Bytes internal SRAM

1. Intro

2. Implementations
   - Blake
   - BlueMidnightWish
   - Grøstl
   - Lane
   - SHAvite3

3. Results

Blake

- HAIFA structure
- Local wide-pipe. State is compressed to digest size at end of each block
- $\frac{1}{4}$ of the state is kept in registers
- One round is derived from stream cipher ChaCha
- Most crucial part is round function G_i, which is called 80 times per message block
- No AES components as building blocks (no SBOX ,MDS matrix).

Note that some constant data is copied from Flash to SRAM for faster access. Therefore it is double counted.

Blake->Constant data in Flash

- 1548 byte code
- 160 byte permutation table
- 32 byte IV
- 64 byte constants
- Overall 1804 byte

Blake->Data in RAM

- 64 byte state
- 64 byte constants
- 32 byte chain value
- 16 byte salt
- 8 byte counter
- 3 byte temporary data
- 64 byte message
- Overall 251 byte

1. Intro

2. Implementations
   - Blake
   - BlueMidnightWish
   - Grøstl
   - Lane
   - SHAvite3

3. Results

BMW

- We implemented the first round version. There are some tweaks for the second round version!
- Wide Pipe design, state is larger than digest
- No AES components as building blocks (no SBOX or MDS matrix at all).
- Initial double pipe is set up on the fly (faster than loading precomputed values)
- Makes extensive use of $xor, >>, <<$ and $+/-mod2^32$, which leads to large code

All constants are directly coded into op-codes.

BMW->Constant data in Flash

- 3558 byte code
- Overall 3558 byte

BMW->Data in RAM

- 64 byte double pipe H
- 128 byte quadruple pipe Q
- 64 byte message
- Overall 266 byte

1. Intro

2. Implementations
   - Blake
   - BlueMidnightWish
   - Grøstl
   - Lane
   - SHAvite3

3. Results

Grøstl

- SP Network with SBOX from AES. Round function very similar to AES.
- Wide Pipe design, state is larger than digest
- only 1/8 of the state is kept in registers
- Final output transformation maps internal state to digest.
- Speed and size optimized versions by Günther A. Roland can be found at http://www.groestl.info/implementations.html

Grøstl->Constant data in Flash

- 4340 byte code
- 256 byte AES SBOX
- 256 byte x-times multiplication table
- Overall 4852 byte

Grøstl->Data in RAM

- 64 byte state
- 64 byte temporary state
- 64 byte chain value
- 8 byte counter
- 64 byte message
- Overall 264 byte

1. Intro
2. Implementations
   - Blake
   - BlueMidnightWish
   - Grøstl
   - Lane
   - SHAvite3
3. Results

Lane

Did not pass to round two. WHY?

- No Wide Pipe design, state is as large as digest
- Half of the state can be kept in registers

Lane->Constant data in Flash

- 1832 byte code
- 256 byte AES SBOX
- Overall 2088 byte

Lane->Data in RAM

- 64 byte (expanded) state
- 32 byte chain value
- 8 byte counter
- 64 byte message
- Overall 264 byte

1. Intro

2. Implementations
   - Blake
   - BlueMidnightWish
   - Grøstl
   - Lane
   - SHAvite3

3. Results

SHAvite3

- Feistel Structure with AES round as building block
- Iteration function is HAIFA
- Narrow Pipe design, state is as large as digest
- Half of the state can be kept in registers

SHAvite3->Constant data in Flash

- 4998 byte code
- 256 byte AES SBOX
- Overall 5254 byte

SHAvite3->Data in RAM

- 64 byte state
- 32 byte chain value
- 8 byte counter
- 32 byte salt
- 32 byte old chain value
- 64 byte message
- Overall 232 byte

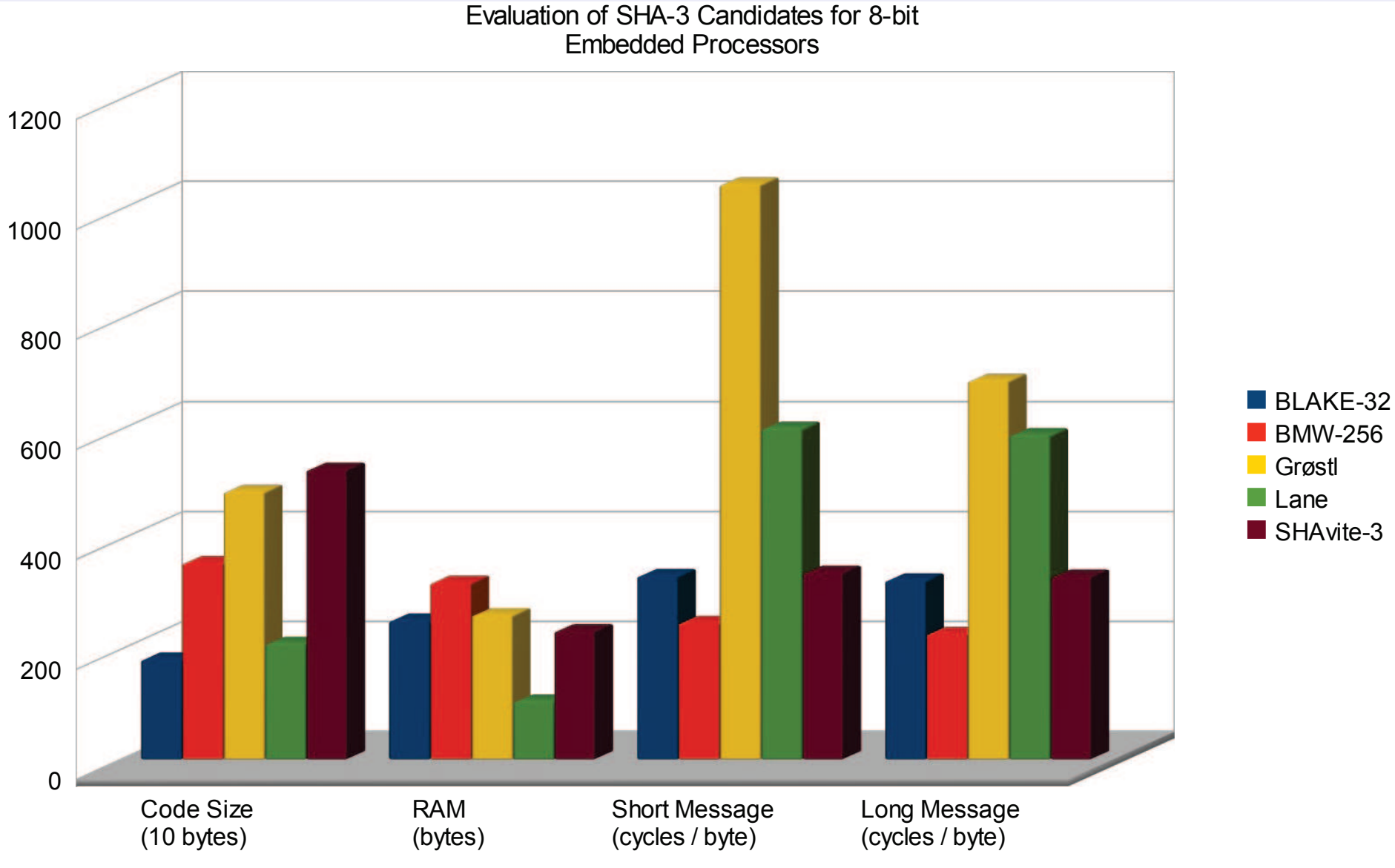
Notes for results

- Again: RAM usages includes 64 byte message buffer
- RAM does not include stack (only a few bytes for IP, because we avoid long push/pop sequences)
- short message: init + padding + hash of one message block + output transformation(if applicable)
- long message: hash of one message block (+output transformation in case of BLAKE, which performs this for every block)

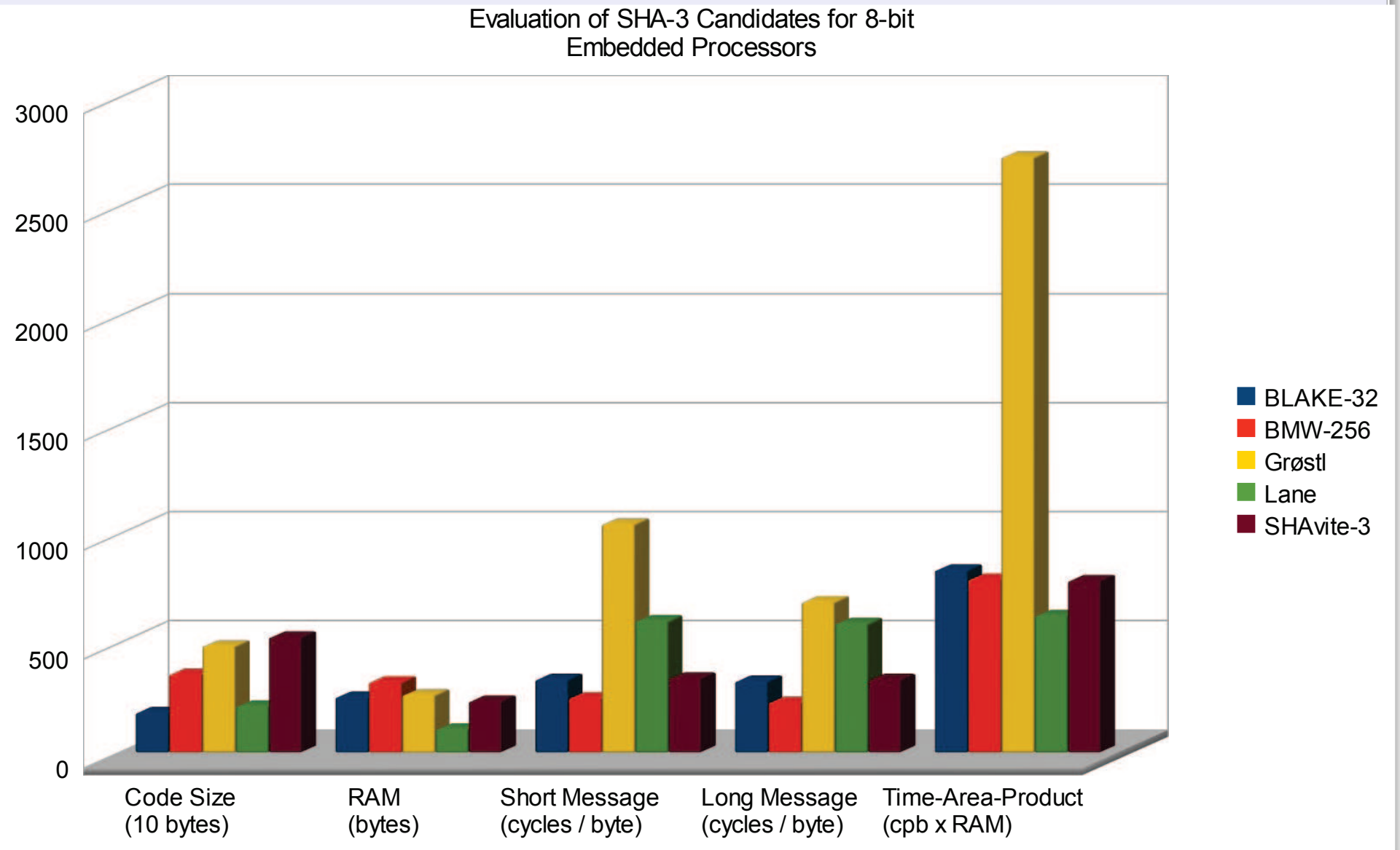
Table: Size and Performance Results

Hashfunction	Flash size	RAM	Short message	Long message
BLAKE-32	1804 byte	251 byte	332 cycles/byte	324 cycles/byte
BMW-256	3558 byte	320 byte	247 cycles/byte	227 cycles/byte
Grøstl-256	4852 byte	264 byte	1044 cycles/byte	687 cycles/byte
Lane	2088 byte	104 byte	600 cycles/byte	588 cycles/byte
SHAvite-3	5254 byte	232 byte	338 cycles/byte	331 cycles/byte

Comparison


Evaluation of SHA-3 Candidates for 8-bit Embedded Processors
1200
1000
800
600
400
200
0
Code Size (10 bytes)
RAM (bytes)
Short Message (cycles / byte)
Long Message (cycles / byte)
BLAKE-32
BMW-256
Grøstl
Lane
SHAvite-3

Comparison


Evaluation of SHA-3 Candidates for 8-bit Embedded Processors
3000
2500
2000
1500
1000
500
0
Code Size (10 bytes)
RAM (bytes)
Short Message (cycles / byte)
Long Message (cycles / byte)
Time-Area-Product (cpb x RAM)
BLAKE-32
BMW-256
Grøstl
Lane
SHAvite-3

End

Sources available at www.schnufff.de (under construction).
Results will be included in XBX soon.

Thanks for your attention.