VLSI Characterization of the Cryptographic Hash Function BLAKE

Luca Henzen, *Student Member, IEEE,* Jean-Philippe Aumasson, Willi Meier, and Raphael C.-W. Phan, *Member, IEEE*

***Abstract*—Cryptographic hash functions are used to protect information integrity and authenticity in a wide range of applications. After the discovery of weaknesses in the current deployed standards, the U.S. Institute of Standards and Technology started a public competition to develop the future standard SHA-3, which will be implemented in a multitude of environments, after its selection in 2012. In this paper, we investigate high-speed and low-area hardware architectures of one of the 14 "second-round" candidates in this competition: BLAKE. VLSI performance results of the proposed high-speed designs indicate a throughput improvement between 16 and 36 % compared to the current standard SHA-2. Additionally, we propose a compact implementation of BLAKE with memory optimization that fits in 0.127 mm^2 of a 0.18 µm CMOS. Measurements reveal a minimal power dissipation of 9.59 µW/MHz at 0.65 V, which suggests that BLAKE is suitable for resource-limited systems.**

***Index Terms*—Cryptographic hash functions, SHA-3, VLSI implementations, low-power, latch memory**

I. Introduction

Hash functions[1] are cryptographic algorithms that take as input a *message* of arbitrary length, and that return a *digest* (or *hash value*) of fixed length (between 160 and 512 bits, in most applications). Hash functions are used in a multitude of protocols, be it for digital signatures within high-end servers, or for authentication of embedded systems.

The research scene of hash functions has seen a surge of works since attacks [1], [2], [3] on the two most deployed hash functions, MD5 and SHA-1. A notable milestone was the forgery of a MD5-signed certificate using a cluster of PlayStation 3's [4]. Such results have led to a lack of confidence in the current U.S. (and de facto worldwide) hash standard, SHA-2 [5], due to its similarity with MD5 and SHA-1. As a response to the potential risks of using SHA-2, the U.S. Institute of Standards and Technology (NIST) has started a public competition—the *NIST Hash Competition*—to develop the future hash standard SHA-3 [6].

SHA-3 is expected to have at least the security of SHA-2, and to achieve this with significantly improved efficiency. By the deadline of October 31, 2008, NIST received 64 submissions, of which 51 were accepted as first round candidates, and 14 as second round candidates in July 2009.

L. Henzen is with the Integrated Systems Laboratory (IIS), ETH Zurich, CH-8092 Zurich, Switzerland (e-mail: henzen@iis.ee.ethz.ch).

J.-Ph. Aumasson is with Nagravision SA, CH-1033 Cheseaux, Switzerland (e-mail: jeanphilippe.aumasson@gmail.com).

W. Meier is with the IAST institute, FHNW, CH-5210 Windisch, Switzerland (e-mail: willi.meier@fhnw.ch).

R.C.-W. Phan is with the Electronic & Electrical Engineering, Loughborough Uni, LE11 3TU, UK (e-mail: r.phan@lboro.ac.uk).

[1]Throughout the paper, "hash functions" refers to cryptographic hash functions, rather than to hash functions used for table lookup.

Besides a sufficient security level, the new hash standard should be implementable on a wide range of environments. In particular, performance in hardware is a crucial criterion to select the future SHA-3, because available hardware is often not flexible or limited, whereas high-end PCs can accommodate a relatively slow function. It is thus necessary to study implementations of candidate algorithms on ASIC and FPGA, and to evaluate their suitability for high-speed or resource-limited environments.

BLAKE [7] is a second round candidate in the NIST Hash Competition. Preliminary analysis suggests that BLAKE performs well in software [8]. In this article, we investigate VLSI implementations of BLAKE, by presenting two architectures for high-speed applications, and reporting on a silicon implementation of a compact BLAKE core. Our work extends the initial hardware evaluation of BLAKE described in its supporting documentation [7], and the subsequent implementations in [9], [10].

The rest of this paper is structured as follows. Section II gives a complete specification of the BLAKE hash function. Section III describes our high-speed architectures and Section IV our compact silicon implementation. Conclusions are drawn in Section V.

II. Algorithm Specification

BLAKE has two main versions: BLAKE-32 and BLAKE-64. This section gives a brief specification of these algorithms. A complete specification can be found in [7].

A. BLAKE-32

The BLAKE-32 algorithm operates on 32-bit words and returns a 256-bit hash value. It is based on the iteration of a *compression function*, described below.

1) Compression Function: Henceforth we shall use the following notations: if m is a message (a bit string), m^i denotes its i-th 16-word block, and m^i_j is the j-th word of the i-th block of m. Indices start from zero, for example a N-block message m is decomposed as $m = m^0 m^1 \dots m^{N-1}$, and the block m^0 is composed of words m^0_0, $m^0_1, m^0_2, \dots, m^0_{15}$. Idem for other bit strings. Endianness conventions are described in [7].

The compression function of BLAKE-32 takes as input four values:

- a chaining value $h = h_0, \dots, h_7$.
- a message block $m = m_0, \dots, m_{15}$.
- a salt $s = s_0, \dots, s_3$.
- a counter $t = t_0, t_1$.

These inputs represent 30 words in total (i.e., 960 bits). The salt is an optional input for special applications, such as randomized hashing [11]. The output of the compression function is a new chaining value $h' = h'_0, \dots, h'_7$ of eight words (i.e., 256 bits). We write

$$h' := \mathsf{compress}(h, m, s, t).$$

The compression function $\mathsf{compress}()$ can be decomposed into three main steps, described in II-A1a) to II-A1c).

a) Initialization: A 16-word internal state $v_0, \dots, v_{15}$ is initialized such that different inputs produce different initial states. This state is represented as a 4×4 matrix:

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \tag{1}$$

The initial state is defined as follows:

$$\begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}, \tag{2}$$

where $c_0, \dots, c_{15}$ are predefined word constants.

b) Round Function: Once the state is initialized, the compression function iterates a series of ten *rounds*. A round is a transformation of the state that computes

$$\begin{array}{ll} \mathsf{G}_0(v_0\ , v_4\ , v_8\ , v_{12}) & \mathsf{G}_1(v_1\ , v_5\ , v_9\ , v_{13}) \\ \mathsf{G}_2(v_2\ , v_6\ , v_{10}, v_{14}) & \mathsf{G}_3(v_3\ , v_7\ , v_{11}, v_{15}) \end{array} \tag{3}$$

and then

$$\begin{array}{ll} \mathsf{G}_4(v_0\ , v_5\ , v_{10}, v_{15}) & \mathsf{G}_5(v_1\ , v_6\ , v_{11}, v_{12}) \\ \mathsf{G}_6(v_2\ , v_7\ , v_8\ , v_{13}) & \mathsf{G}_7(v_3\ , v_4\ , v_9\ , v_{14}) \end{array} \tag{4}$$

where, at round r, $\mathsf{G}_i(a, b, c, d)$ sets

$$\begin{array}{rcl} a & := & a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\ d & := & (d \oplus a) \ggg 16 \\ c & := & c + d \\ b & := & (b \oplus c) \ggg 12 \\ a & := & a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\ d & := & (d \oplus a) \ggg 8 \\ c & := & c + d \\ b & := & (b \oplus c) \ggg 7 \end{array} \tag{5}$$

The G function[2] uses ten permutations of $\{0, \dots, 15\}$, written $\sigma_0, \dots, \sigma_9$, which are fixed by the design. G also uses the constants $c_0, \dots, c_{15}$. The unary operator $\ggg$ denotes rotation of words towards least significant bits.

Note that the first four calls $\mathsf{G}_0, \dots, \mathsf{G}_3$ in (3) can be computed in parallel, because each updates a distinct column of the state. The sequence $\mathsf{G}_0, \dots, \mathsf{G}_3$ is called a *column step*. Similarly, the last four calls $\mathsf{G}_4, \dots, \mathsf{G}_7$ in (4) update distinct diagonals and are called a *diagonal step*.

c) Finalization: After the sequence of rounds, the new chaining value h' is extracted from the state $v_0, \dots, v_{15}$ with input of the initial chaining value h and the salt s:

$$\begin{array}{rcl} h'_0 & := & h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\ h'_1 & := & h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\ h'_2 & := & h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\ h'_3 & := & h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\ h'_4 & := & h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\ h'_5 & := & h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\ h'_6 & := & h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\ h'_7 & := & h_7 \oplus s_3 \oplus v_7 \oplus v_{15} \end{array} \tag{6}$$

2) Hashing a Message: When hashing a message, the function starts from an *initial value* (IV), and the iterated hash process computes intermediate hash values that are called *chaining values*. Before being processed, a message is first *padded* so that its length is a multiple of the block size (512 bits). It is then processed block per block by the compression function, as described below:

```
$h^0 :=$ IV
**for** $i = 0, \dots, N-1$
    $h^{i+1} := \mathsf{compress}(h^i, m^i, s, \ell^i)$
**return** $h^N$
```

Here, ℓ^i is the number of message bits in $m^0, \dots, m^i$, that is, excluding the bits added by the padding. It is used to avoid certain generic attacks on the iterated hash (e.g., [12]). The salt s is chosen by the user, and set to zero by default.

B. BLAKE-64

BLAKE-64 operates on 64-bit words and returns a 512-bit hash value. All lengths of variables are doubled compared to BLAKE-32: for instance, chaining values are 512-bit, message blocks are 1024-bit, salt is 256-bit, counter is 128-bit.

The compression function of BLAKE-64 is similar to that of BLAKE-32 except that it makes 14 rounds instead of ten, and that $\mathsf{G}_i(a, b, c, d)$ uses rotation distances 32, 25, 16, and 11, respectively. After ten rounds, the round function uses the permutations $\sigma_0, \dots, \sigma_4$ for the last four rounds. The algorithm for hashing a message is similar to that of BLAKE-32.

III. High-Speed VLSI Implementations

In this section we investigate high-speed implementations of BLAKE, with an iterative decomposition of the round process.

Different architectures are made possible by varying the number of integrated G modules. Modern high-speed communication systems where the space is not a fierce constraint can take advantage of architectures with eight G modules or even with a complete round-unrolled circuit [13]. At the opposite, by scaling the number of G modules the design becomes slower but decreases in size (see design proposals of [7]).

Besides the round computation, BLAKE requires some circuitry to perform initialization and finalization; for instance,

[2]In the following, for statements that do not depend on the index i we shall omit the subscript and write simply G.

32 w-bit XORs are required to compute (2) and (6), where $w = 32$ for BLAKE-32 and $w = 64$ for BLAKE-64. Furthermore, the complete execution of initialization and finalization can be performed in the same clock cycle, when the new message block is given. Like most hash functions, BLAKE uses some constant values, which are

- the initial value IV_i (eight w-bit words);
- the 16 round constants c_i;
- the ten permutations σ_i (in total of 640 bits).

These values are used mainly by the G function; the best solution is to hard-code them without using special macro blocks for storage. Since BLAKE iterates a series of rounds over an internal state, additional sequential components are required to store the following 44 values:

- the 8-word chaining value h;
- the 16-word internal state v ;
- the 4-word of the salt value s;
- the 16-word message block m.

The two words of the counter t need not be stored. In high-speed architectures, the initialization process (the only phase where the counter is used) is indeed executed in a single clock cycle. Moreover, we decided to take the counter externally as input together with the message block. This choice is motivated by the fact that the counter during the last call of the compression function “knows” the number of padded bits inside the last message block. It is thus natural to treat it like a normal input. The sequential area is thus made up by $44\times w$ registers (i.e., 1408 for BLAKE-32, 2816 for BLAKE-64) plus some additional registers for the control unit.

To exploit the full parallelizability of BLAKE, two types of design have been coded in VHDL. Referring to [14], [7], the first is called [8G], which corresponds to a straightforward round-iterative implementation with eight G modules computing the column and diagonal step; and the second, called [4G], where only four parallel G modules concurrently compute the two steps. Outside the round module, the sequential part (register memories), and the components for initialization and finalization, we added a control unit, based on a simple finite-state machine, which computes the round increment and starts or terminates the hashing process. Fig. 1 shows a block diagram of the [8G]- and [4G]-BLAKE cores. During the round iteration, only the state memory and the [8G], respectively [4G], module are mainly involved.

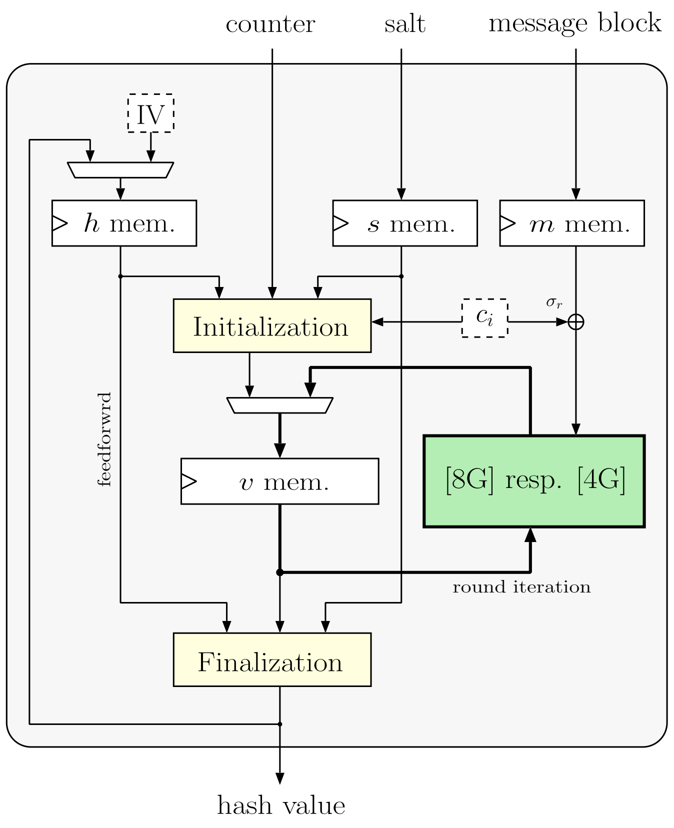

Fig. 1. The main architecture of the [8G]- and [4G]-BLAKE cores.

A. Round Rescheduling

The G function of BLAKE is a modified version of the core function of the stream cipher ChaCha [15] proposed by Bernstein in the context of the eSTREAM Project[3]. Speed limits for plain designs implementing several architectures of ChaCha have been reported in [14]. The introduction of the addition with the message/constant (MC) -pair in the G function leads to an increment of the propagation delay. If in the core function (similar to G) the maximal delay is given by the total delay of four XORs and four modular adders (rotation is a simple re-routing of the word without effective propagation delay), the slightly modified G function inserts an addition with the MC-pair. Accordingly, the maximal frequency values of analogous BLAKE architectures (cf. [7]) are slightly lower than those obtained for the stream cipher ChaCha. However, with a rescheduling of the G computation, it is possible to recover the original maximal path of ChaCha (four XORs and four adders), hence decreasing the overall propagation delay of the core function. Observing the flow dependencies in (5), it is clear that the addition with the MC-pair is independent (message word and constant are unrelated to the state v) and can be computed in parallel to the other computations. If in a single call of G, similarly to the core function of ChaCha, each update of the state has been conceived to operate sequentially, the MC-pair addition can be shifted within the computations. It is thus possible to anticipate it, reducing the critical path of G. The rescheduled $\mathsf{G}_i(a^*, b, c, d)$ computes

$$\begin{array}{rcl} a & := & a^* + b \\ d & := & (d \oplus a) \ggg r_0 \\ c & := & c + d \\ b & := & (b \oplus c) \ggg r_1 \\ a & := & a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\ d & := & (d \oplus a) \ggg r_2 \\ c & := & c + d \\ b & := & (b \oplus c) \ggg r_3 \\ a* & := & a + (m_{\sigma_{r+1}(2i)} \oplus c_{\sigma_{r+1}(2i+1)}) \end{array} \qquad (7)$$

[3]Organized by the European NoE ECRYPT, the eSTREAM Project was a multi-year effort running from 2004 to 2008, which identified a portfolio of promising stream ciphers

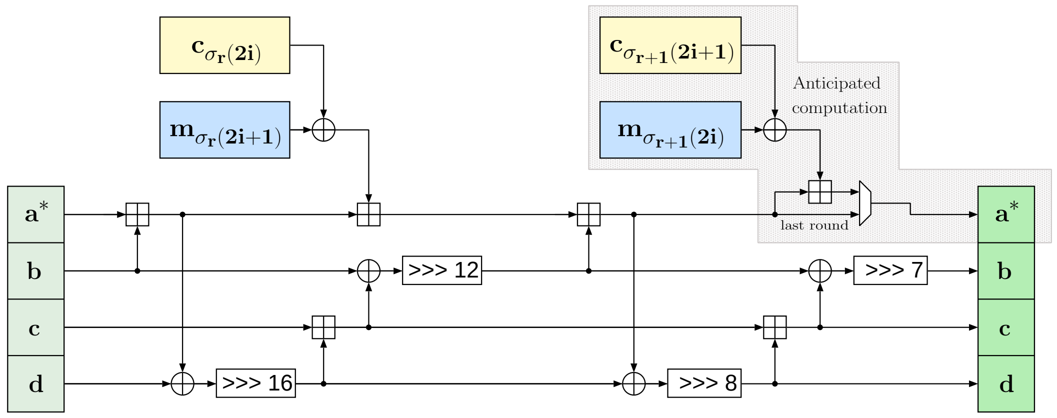
$\mathbf{c}_{\sigma_\mathbf{r}(\mathbf{2i})}$
$\mathbf{m}_{\sigma_\mathbf{r}(\mathbf{2i+1})}$
$\mathbf{c}_{\sigma_\mathbf{r+1}(\mathbf{2i+1})}$
$\mathbf{m}_{\sigma_\mathbf{r+1}(\mathbf{2i})}$
Anticipated computation
last round
$\mathbf{a}^*$
b
c
d
>>> 12
>>> 16
>>> 7
>>> 8
$\mathbf{a}^*$
b
c
d

Fig. 2. Block diagram of the rescheduled G function. Note: the round index of the second message/constant pair is increased by one.

TABLE I
PERFORMANCE COMPARISON FOR A 0.18 μM CMOS TECHNOLOGY.

Algorithm	Area [kGE]	Cyc.	Freq. [MHz]	Thr. [Gbps]	HW-Eff. [kbps/GE]
[4G]-BLAKE-32	48	21	240	5.847	123
[4G]-BLAKE-64	98	29	204	7.192	74
[8G]-BLAKE-32	79	11	137	6.376	81
[8G]-BLAKE-64	147	15	106	7.216	49
BLAKE-32[a] [9]	46	22	171	3.971	87
BMW-256 [9]	170	1	10	5.385	32
CH16/32(-256)[b] [9]	59	8	146	4.665	79
ECHO-256 [9]	141	97	142	2.246	16
Fugue-256 [9]	46	2	256	4.092	88
Grøstl-256 [9]	58	22	270	6.290	108
Grøstl-512 [16]	340	14	85	6.225	18
Hamsi-256 [9]	59	1	174	5.565	95
JH-256 [9]	59	39	380	4.992	85
Keccak(-256) [9]	56	25	488	21.229	377
Luffa-256 [9]	45	9	483	13.741	306
Shabal-256 [9]	54	50	321	3.282	61
SHAvite-3-256 [9]	57	37	228	3.152	55
SIMD-256 [9]	104	36	65	0.924	9
Skein-256-256 [9]	59	10	74	1.882	32
Skein-512-512 [9]	102	10	49	2.205	22
SHA-256 [9]	19	66	302	2.344	122
SHA-512 [17]	31	88	169	1.969	64

[a] Salt support is omitted.
[b] We refer to the CubeHash candidate [18].

TABLE II
PERFORMANCE COMPARISON FOR A 0.13 μM CMOS TECHNOLOGY.

Algorithm	Area [kGE]	Cyc.	Freq. [MHz]	Thr. [Gbps]	HW-Eff. [kbps/GE]
[4G]-BLAKE-32	43	21	330	8.047	187
[4G]-BLAKE-64	92	29	291	10.265	111
[8G]-BLAKE-32	67	11	201	9.365	140
[8G]-BLAKE-64	139	15	158	10.802	78
CH16/32 [19]	34	16	578	9.248	269
ECHO-256 [20]	521	9	87	14.850	29
ECHO-512 [20]	517	11	83	7.750	15
Hamsi-256 [21]	22	7	1 080	4.937	224
Hamsi-512 [21]	50	13	820	4.036	81
Keccak [22]	48	18	526	29.900	623
Luffa-256 [23]	27	9	444	12.642	471
Luffa-512 [23]	44	8	444	12.642	286
Shabal [19]	41	52	645	6.351	154
SHA-256 [24]	22	68	794	5.975	271
SHA-512 [24]	43	84	746	9.096	210

TABLE III
PERFORMANCE COMPARISON FOR A 90 NM CMOS TECHNOLOGY.

Algorithm	Area [kGE]	Cycles	Freq. [MHz]	Thr. [Gbps]	HW-Eff. [kbps/GE]
[4G]-BLAKE-32	38	21	621	15.143	396
[4G]-BLAKE-64	79	29	532	18.782	237
[8G]-BLAKE-32	65	11	376	17.498	269
[8G]-BLAKE-64	128	15	298	20.317	158
Fugue-256[25]	110	2	870	13.913	127

where r_i are the rotation indices for BLAKE-32 and BLAKE-64, and a^* corresponds to the modified first input/output variable after the MC addition. Fig. 2 shows the block diagram of the modified G function. To keep the correct functional behavior, a 2-input MUX should be inserted before the sequential logic, hence allowing the record of a instead of a^* in the last round.

B. Performance Analysis

To evaluate the speed-up provided by the G rescheduling, we coded the [8G] and [4G] architectures in VHDL and we synthesized them for BLAKE-32 and BLAKE-64 with the Synopsys Compiler. Our results refer to fully-autonomous designs, which take as input salt, counter, and message blocks and generate the final hash value. Moreover, to obtain an exhaustive analysis of the BLAKE hash cores, the designs have been synthesized in four different UMC technologies: 0.18 μm, 0.13 μm, and 90 nm.

Tab. I-III present a detailed performance comparison with the current standard SHA-2, and with other second round candidates in the NIST Hash Competition for which performance figures are available. Each entry refers to a post-synthesis implementation, and the last column reports the hardware efficiency, i.e., the ratio between throughput and required area. Only for 0.18 μm we were able to provide a full comparison between the 14 candidates. This was possible thanks to the results provided in [9]. Fig. 3 illustrates the trade-off between area and processing time for the 256-bit versions of the candidate functions plus the SHA-2 standard. Note that our designs for BLAKE-32 support salted hashing which is not the case in [9].

Compared to the architectures presented in [7], we obtain a 20 % speed-up, due to the delay reduction of the round rescheduling process described in the previous section. We should however take into account an area increase caused by the integration of the register-based memories for message

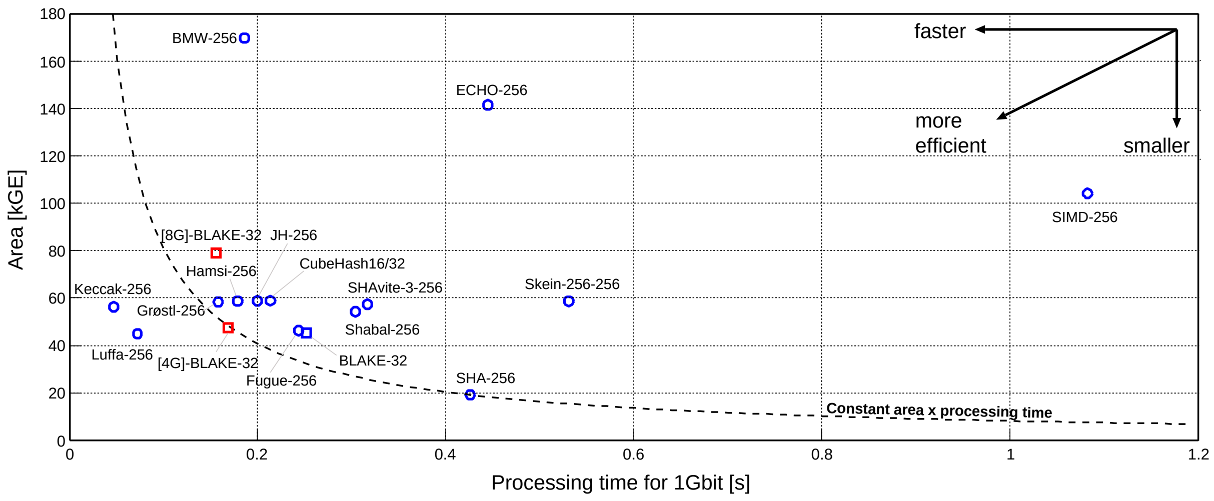

Fig. 3. Processing time for 1 Gbit of data versus total area of the 14 second round candidates (0.18 µm CMOS technology). The blue points refer to the implementations of [9]. The BLAKE-32 cores presented here are in red. The dashed lines defines the limit of constant efficiency equal to the SHA-2 core.

block, chaining value, and salt; note that the previous designs of [7] represent only the compression function.

Comparing the proposed BLAKE cores with the SHA-2 family, we observe a substantial throughput gain. This improvement comes at the cost of an area increase, which can also be a side effect of the alleged security improvement. Comparing with the other candidates, BLAKE is faster than about half of them. If we take into account that the function Blue Midnight Wish [26] requires a large area to achieve the same speed, we could assert that our architectures improve the results of [9], outperforming in efficiency a set of four candidate algorithms with similar throughput performances, i.e., Grøstl [16], Hamsi [21], JH [27], and CubeHash [18] (see Fig. 3). With the application of the round rescheduling, we could indeed increase the hardware efficiency up to the value achieved by SHA-256 in 0.18 µm.

The functions Keccak [22] and Luffa [23] outperform every candidate in maximal achievable speed, requiring at the same time limited-area hardware. This mainly follows from their sole use of Boolean operators, rather than of modular additions. Note however that such optimization for hardware comes at a price in terms of performance in software (where the function cannot benefit of CPU's arithmetic instructions). Moreover, previous cryptanalysis results suggest that such designs may have structural flaws [28], [29].

IV. Silicon Implementation of a Compact BLAKE-32 Core

We designed a compact architecture of BLAKE-32, to satisfy the stringent restrictions of resource-constrained environments. Besides an area reduction, the cryptographic core must also keep the energy dissipation at minimal values. Following these two design principles, we concentrated our efforts in the reduction of the round circuit and in the implementation of efficient memory modules (see Sec. IV-A).

As previously noted, BLAKE relies on eight calls of the G function within the column and diagonal steps. Inside the G function, the computation that requires most of the area resources is the modular addition. Instead of implementing four G modules with six independent 32-bit adders, we opted for a single adder, where the G function is iteratively decomposed in ten steps. This causes an increase of the *per-message block* processing time, but contributes to a limited overall size. Fig. 4 shows the block diagram of the proposed compact architecture. For the G computation, two 32-bit XOR gates and a rotation selector (r_i defines the different rotation numbers) are implemented in conjunction with the 32-bit adder (cf. ② in Fig. 4). Each variable required by the hashing process is stored in optimized two-port memories. In total, five memory elements are needed, while an intermediate 32-bit register allows the extraction of temporary state words. This architecture leads to a total latency in clock cycles of 816 for 512-bit message block. In addition to the 10$\times$ 8$\times$10 cycles to complete the round function, 16 cycles are indeed needed for the initialization process. Moreover, the initialization is started while the update of the chaining value (finalization) is still ongoing. Here after sorting the selected state word v_i (required for the h' computation, cf. ③ in Fig. 4), the free memory slot is filled with the new chaining value or with the result coming from ①, respectively.

If the output of the architecture is the 32-bit value stored in the intermediate register, the input is a 32-bit word which is consequently routed to the memories for message block, salt, or block counter.

A. Memory Architecture

The VLSI implementation of BLAKE-32 needs memory to store 16 words of internal state and eight words of chaining value, plus additional registers to store the salt (four words), the counter (two words), and the message block (16 words), i.e., in total 1472 bits of memory. The counter is used during four clock cycles and needs thus to be stored. Compared to the minimum circuit needed to implement the compression function (initialization, rounds, and finalization), the memory units is the main contribution in terms of area and energy consumption. It is thus of primary interest to design special-purpose register elements, to decrease the global resource

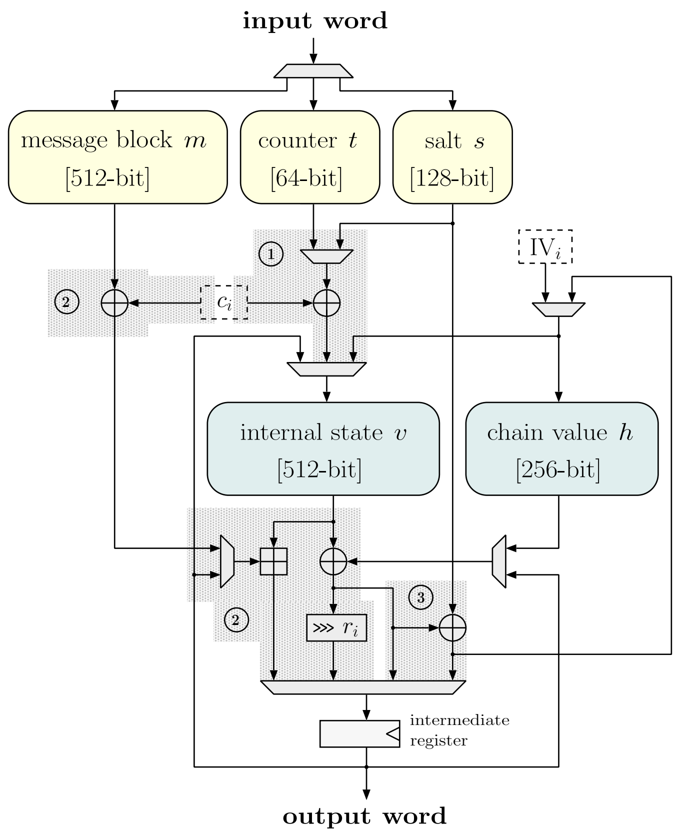

Fig. 4. Block diagram of the implemented lightweight BLAKE-32. All connections are 32-bit wide.

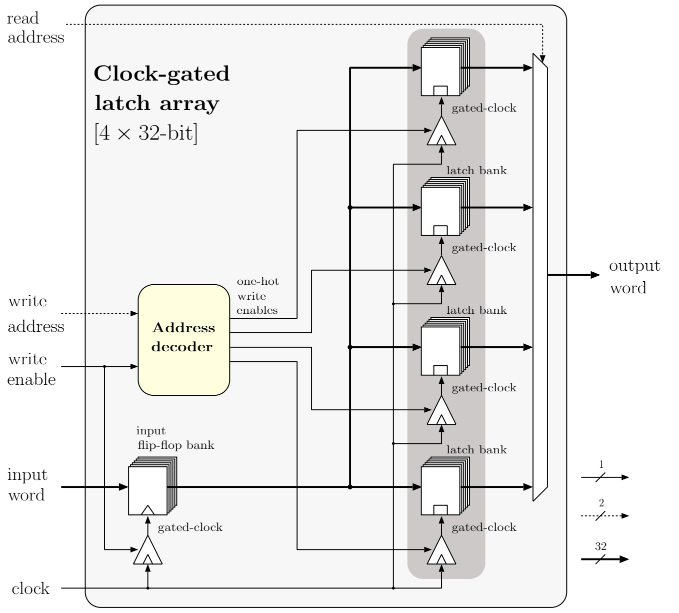

Fig. 5. Overview of a 4-word memory unit implemented as clock-gated latch array.

TABLE IV
SIZE COMPARISON IN GE[a] OF THE MEMORY ELEMENTS IN A 0.18 µM CMOS TECHNOLOGY (POST LAYOUT RESULTS FOR 200 MHZ) .

Word number	2	4	8	16
Standard FF	585	1234	2370	5434
Latch array	550	926	1681	3376
Area gain [%]	6	25	29	39

[a] One GE corresponds to the area of a two-input drive-one NAND gate in the 0.18 µm CMOS technology (area 9.3744 µm^2).

requirements of the hash core. We introduced in the compact architecture of BLAKE-32 semi-custom memories based on clock-gated latch arrays, able to store at most one word per cycle. In general, depending on the word number of the target value to be stored, these memories replace the standard flip-flop gates by latch gates. The latches are organized in 32-bit banks, so that each bank stores a single word and is triggered by a dedicated gated clock [30].

Our example architecture in Fig. 5 depicts a 4-word latch array, that is used to store the salt value. In the address decoder, the different one-hot enable signals are activated, depending on the write address. To prevent timing loops inside the logic, caused by the transparent behavior of latches, an input flip-flop bank is added. This bank is in turn driven by a gated clock generated with the write enable signal, while the outputs of the flip-flops are connected to the inputs of all latch banks. When a write enable occurs, the input word is firstly stored inside the flip-flop bank and subsequently passed to the activated latch bank.

To keep the functionality similar to a normal flip-flop-based memory, the outputs of the latch array are connected with a large multiplexer driven by the read address signal. This allows an instantaneous response of the memory.

An area comparison between the proposed latch-array and standard flip-flop-based memories is shown in Tab. IV. By decreasing the number of words, the two memories get closer in size. However, we still achieve a slight area saving, even with the smaller used memory size (counter). As can be seen in Fig. 4, the compact BLAKE-32 architecture works with five memories. This means an overall area reduction of about 34 % for the memory components and 28 % for the complete design. In order to compare the power consumption, we integrated two equal BLAKE-32 cores up to layout, using different memory strategies. With the aid of post place & route power simulations, a 60 % mean reduction in the energy dissipation of the five latch-based memory components has been measured, leading to a global energy decrease of 36 % for the complete design.

B. ASIC Constraints

The compact BLAKE-32 architecture has been coded in VHDL and synthesized with the Synopsys Compiler using the UMC 1P/6M 0.18 µm technology. The Cadence SoC Encounter System has then been used to place & route the final layout of the ASIC. The chip layout and the die photo are presented in Fig. 6. The BLAKE-32 core fills only 0.127 mm^2, which is only a small fraction of the total chip (size 1.565 mm×1.565 mm, i.e., 2.450 mm^2). Tab. V gives an overview of the area partitioning of the hash core. In the remaining space, additional projects distinct from BLAKE have been integrated.

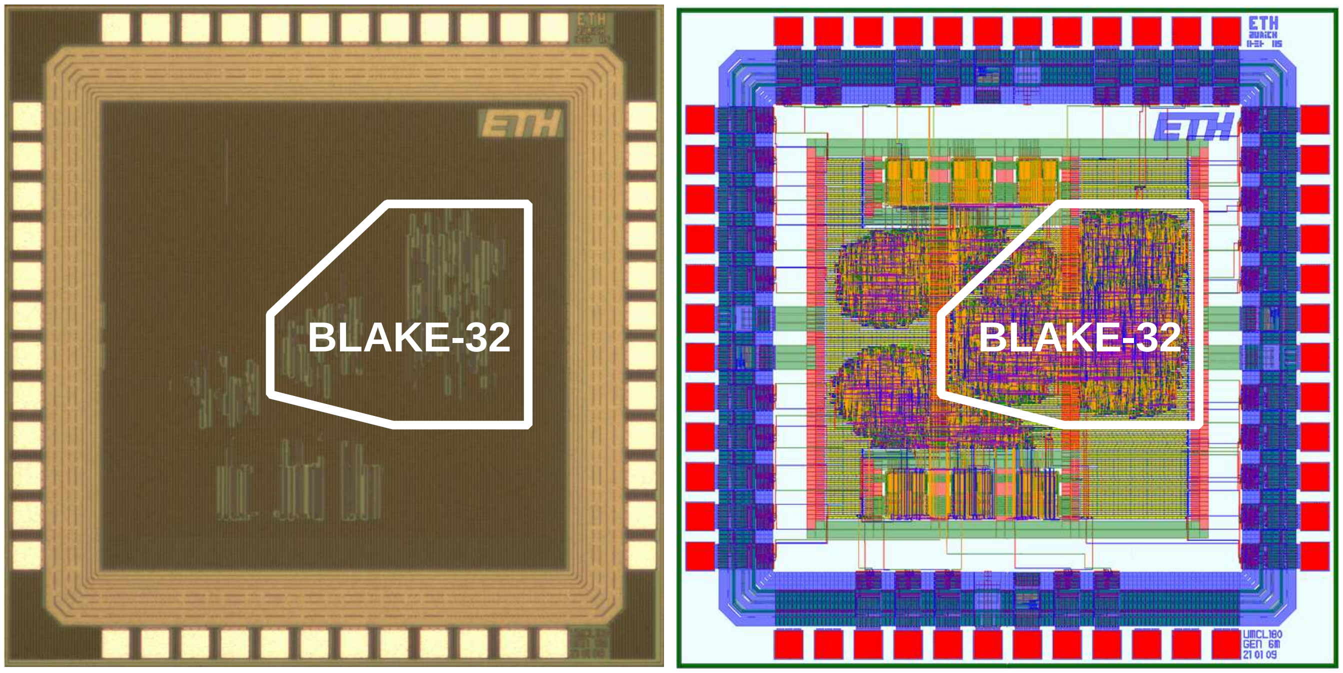

Fig. 6. Die photo (left) and layout (right) of the compact BLAKE-32 implementation in 0.18 µm CMOS technology. Note that the ASIC hosts some additional unrelated circuitry for other cryptographic algorithms.

TABLE V
DETAILED CHIP AREA AND POWER CONTRIBUTION OF THE COMPACT BLAKE-32 CORE.

Component	Area [GE]	Area [%]	Power[a] [%]
m mem. (16 w)	3295	24.3	3.4
v mem. (16 w)	3457	25.5	26.4
h mem. (8 w)	1681	12.4	3.1
s mem. (4 w)	926	6.8	0.4
t mem. (2 w)	550	4.1	0.2
Controller	776	5.7	6.6
Round	2890	21.3	60.0
Total	13 575	100.0	100.0

[a] The power consumption values of the single modules are extracted from a post-layout simulation-based power analysis.

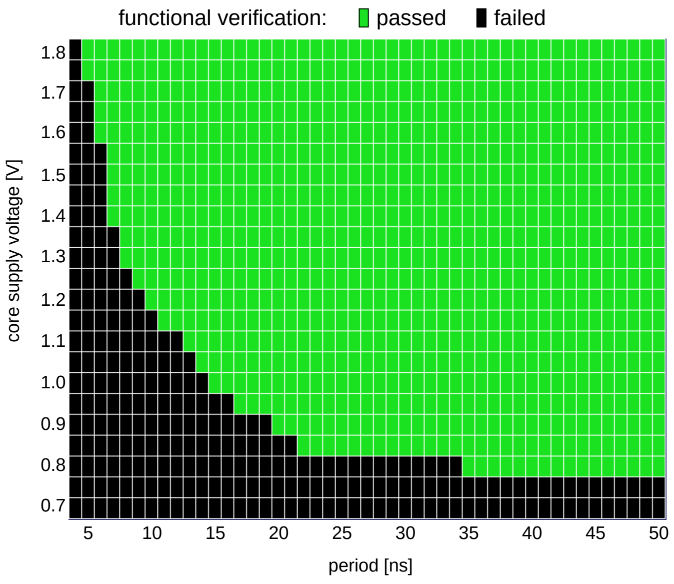

Fig. 7. Period-voltage shmoo plot for the compact BLAKE-32 design.

C. Measurements and Performance Comparison

To test the correct functional behavior, the fabricated chip has been stimulated using a HP83000 digital tester, under different setups and stimuli vectors. The characteristic period vs. supply voltage shmoo plot is presented in Fig. 7. The evident aspect is that the maximal working frequency strongly depends on the supply voltage.

To reach 200 MHz the chip must be supplied with the technology nominal voltage of 1.8 V. With these parameters, post layout power simulations have been performed, in order to evaluate the single energy contributions of the chip components (cf. last column of Tab. V). Memory modules, sparsely used during the compression process, consume less energy independently from their size. This is the primary goal of the proposed memory architecture. The m memory, which is one of the largest memory units, but updated only once per compression, dissipates indeed the same amount of power like the half-sized h memory . This leads to a minor global contribution by the storing elements, which consume globally only 33.5 % of the total power, even if they fill more than 70 % of the chip area.

In resource-constrained environments like RFID systems or smart cards, power is often limited, like the total silicon size. The decrease of the supply voltage becomes an efficient solution to reduce the overall consumption. As can be seen from Fig. 7, this causes a proportional slowdown of the working frequency. It becomes thus important that in low voltage regimes the frequency still satisfies the speed requirements of the target communication protocol. For the case of the RFID standards ISO 18000, ISO 14443, or ISO 15693, working in high-frequency (HF) and low-frequency (LF) domains, the

TABLE VI
PERFORMANCE COMPARISON OF DIFFERENT CRYPTOGRAPHIC PRIMITIVES.

Algorithm	Area [GE]	Latency [cycles]	I_{mean} [µA]	Tech. [µm]
BLAKE-32	13.575	816	0.7	0.18
SHA-1 [33]	6.122	344	7.7	0.18
SHA-256 [34]	10.868	1128	3.2	0.35
MD5 [34]	8.001	712	3.2	0.35
AES-128 [35]	3.400	1032	3.0	0.35
ECC-163 [36]	11.904	306 000	5.7	0.18

operating frequency could reach the 13.56 MHz [31], [32]. By selecting a correct functional region from the shmoo plot, we could decrease the supply voltage at 0.65 V, ensuring a correct behavior of the BLAKE-32 core up to 18 MHz.

Real power measurements of the core energy dissipation have been performed using a long randomized message as input. The mean power consumption, measured during the compression process, indicates that the chip dissipates 22.32 mW in nominal condition at the working frequency of 200 MHz. For the case of 13.56 MHz, i.e., the maximal frequency of HF RFID applications, the core dissipates 130 µW at 0.65 V, which is far below the predictions given in [37] (<500 µW). However, to meet the restrictive constraints given in [38] (mean current below 10 µA), the frequency should be scaled to 100 kHz (see [39]). At this speed the chip requires only 0.55 V to generate correct output data. Tab. VI illustrates a comparison with other cryptographic protocols (not necessarily hash functions, and of different security levels), e.g., AES-128, at this target frequency. Although the area is the largest, the BLAKE core turns out to be the most efficient circuit in terms of mean current. Nonetheless, the area demand of the proposed implementation could be further reduced by removing the message block memory and the salt support. For the first case we could suppose the presence of an external tamper-resistant memory, that stores the secret message, for the second case we simply omit an added functionality of the BLAKE algorithm. We designed up to layout a modified version of the compact BLAKE core. The size of this reduced BLAKE-32 version requires just 8.802 kGE. In Tab. VII, a comparison with other compact implementations of second round candidates of the SHA-3 competition is proposed. The results demonstrate a fair optimal trade-off between area and speed for our compact BLAKE designs, which are well-suited for area-limited embedded systems.

V. CONCLUSION

The future cryptographic hash standard SHA-3 should be suitable and flexible for a wide range of applications, featuring at the same time an optimal security strength. In this work, we presented a complete hardware characterization of the BLAKE candidate, using different design approaches to generate fully-autonomous high-speed and compact implementations. A round rescheduling technique and a special-purpose memory design are also proposed. Post-synthesis results of speed optimized architectures demonstrate a throughput improvement of up to 36 % for 256-bit hashing and up to 16 % for 512-bit compared to iterative bounded implementations of the current standard SHA-2. Furthermore, a low-power compact implementation of BLAKE-32 has been fabricated in a 0.18 µm CMOS. Measurements reveal a minimal power dissipation of 130 µW at the RFID nominal frequency of 13.56 MHz. We believe that a similar memory approach for compact VLSI implementations of cryptographic protocols is a valuable choice to reduce the area and power consumption of the integrated circuit.

The wide spectrum of achieved performances paves the way for the application of the BLAKE function to various hardware implementations.

TABLE VII
OVERVIEW OF LOW-AREA ARCHITECTURES OF THE SHA-3 ROUND 2 CANDIDATES.

Algorithm	Area [kGE]	Freq. [MHz]	Thr. [Mbps]	Tech. [µm]
BLAKE-32	13.575	215	135.0	0.18
BLAKE-32[a]	8.602	100	62.7	0.18
BLAKE-32 [10]	25.569	31	15.4	0.35
Grøstl [10]	14.622	56	145.9	0.35
Keccak[b] [22]	5.000	200	52.9	0.13
Luffa-256 [23]	10.157	100	28.7	0.13
Skein-256 [10]	12.890	80	19.8	0.35

[a] This compact core uses an external memory to hold the message block and does not provide salted hashing.

[b] This implementation uses external memory to hold 1600-bit intermediate values during the hashing of a message.

ACKNOWLEDGMENT

The authors would like to thank F. Carbognani for his support during the VLSI design and P. Meinerzhagen for his valuable effort in the memory analysis.

REFERENCES

[1] X. Wang and H. Yu, “How to break MD5 and other hash functions,” in *Advances in Cryptology - EUROCRYPT 2005*, ser. Lecture Notes in Computer Science, vol. 3494. Springer Berlin / Heidelberg, 2005, pp. 19–35.

[2] C. D. Cannière and C. Rechberger, “Finding SHA-1 characteristics: General results and applications,” in *Advances in Cryptology - ASIACRYPT 2006*, ser. Lecture Notes in Computer Science, vol. 4284. Springer Berlin / Heidelberg, 2006, pp. 1–20.

[3] M. Stevens, A. Lenstra, and B. de Weger, “Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities,” in *Advances in Cryptology - EUROCRYPT 2007*, ser. Lecture Notes in Computer Science, vol. 4515. Springer Berlin / Heidelberg, 2007, pp. 1–22.

[4] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger, “MD5 considered harmful today. Creating a rogue CA certificate,” in *Proc. of the 25st Chaos Communication Congress*, 2008.

[5] NIST, “Announcing the secure hash standard,” FIPS 180-2, Technical report, 2002.

[6] ——, “Call for a new cryptographic hash algorithm (SHA-3) family,” Federal Register, Vol.72, No.212, 2007, http://www.nist.gov/hash-competition.

[7] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, “SHA-3 proposal BLAKE,” Submission to NIST, 2008, http://131002.net/blake/.

[8] D. J. Bernstain and T. L. (editors), “eBASH: ECRYPT benchmarking of all submitted hashes,” http://bench.cr.yp.to.

[9] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, and A. Szekely, “High-speed hardware implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein,” Cryptology ePrint Archive, Report 2009/510, 2009.

[10] S. Tillich, M. Feldhofer, W. Issovits, T. Kern, H. Kureck, M. Mühlberghuber, G. Neubauer, A. Reiter, A. Köfler, and M. Mayrhofer, "Compact hardware implementations of the SHA-3 candidates ARIRANG, BLAKE, Grøstl, and Skein," Cryptology ePrint Archive: Report 2009/349, 2009.

[11] NIST, "SP 800-106, randomized hashing digital signatures," 2007.

[12] J. Kelsey and B. Schneier, "Second preimages on n-bit hash functions for much less than 2^n work," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, R. Cramer, Ed., vol. 3494. Springer, 2005, pp. 474–490.

[13] R. Lien, T. Grembowski, and K. Gaj, "A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512," in *Topics in Cryptology - CT-RSA 2004*, ser. Lecture Notes in Computer Science, vol. 2964. Springer Berlin / Heidelberg, 2004.

[14] L. Henzen, F. Carbognani, N. Felber, and W. Fichtner, "VLSI hardware evaluation of the stream ciphers Salsa20 and ChaCha, and the compression function Rumba," in *Proc. of the IEEE Int. Conference on Signals, Circuits and Systems (SCS)*, Nov. 2008, pp. 1–5.

[15] D. J. Bernstein, "ChaCha, a variant of Salsa20," 2007, http://cr.yp.to/chacha.html.

[16] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, "Grøstl - a SHA-3 candidate," Submission to NIST, 2008, http://www.groestl.info.

[17] A. Satoh, "ASIC hardware implementations for 512-bit hash function Whirlpool," in *Proc. of the IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Seattle, WA, May 2008, pp. 2917–2920.

[18] D. J. Bernstein, "CubeHash specication (2.b.1)," Submission to NIST, 2008, http://cubehash.cr.yp.to/.

[19] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, and W. Fichtner, "Hardware implementations of the SHA-3 candidates Shabal and CubeHash," in *Proc. of the IEEE Midwest Symposium on Circuits and Systems (MWSCAS)*, Cancun, Mexico, Aug. 2009, pp. 515–518.

[20] L. Lu, M. O'Neill, and E. Swartzlander, "Hardware evaluation of SHA-3 hash function candidate ECHO," in *Proc. of the Claude Shannon Workshop on Coding and Cryptography*, 2009.

[21] O. Küçük, "The hash function Hamsi," Submission to NIST, 2008, http://homes.esat.kuleuven.be/~okucuk/hamsi/.

[22] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family," Submission to NIST, 2008, http://keccak.noekeon.org/.

[23] C. D. Canniere, H. Sato, and D. Watanabe, "Hash function Luffa," Submission to NIST, 2008, http://www.sdl.hitachi.co.jp/crypto/luffa/.

[24] Y. K. Lee, H. Chan, and I. Verbauwhede, "Iteration bound analysis and throughput optimum architecture of SHA-256 (384,512) for hardware implementations," in *Information Security Applications*, ser. Lecture Notes in Computer Science, vol. 4867. Springer Berlin / Heidelberg, 2008, pp. 102–114.

[25] S. Halevi, W. E. Hall, and C. S. Jutla, "The hash function Fugue," Submission to NIST, 2008, http://domino.research.ibm.com/comm/research_projects.nsf/pages/fugue.index.html.

[26] D. Gligoroski, V. Klima, S. J. Knapskog, M. El-Hadedy, J. Amundsen, and S. F. Mjølsnes, "Cryptographic hash function Blue Midnight Wish," Submission to NIST, 2008, http://www.q2s.ntnu.no/blue_midnight_wish/start.

[27] H. Wu, "The hash function JH," Submission to NIST, 2008, http://icsd.i2r.a-star.edu.sg/staff/hongjun/jh/.

[28] I. Dinur and A. Shamir, "Cube attacks on tweakable black box polynomials," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, A. Joux, Ed., vol. 5479. Springer, 2009, pp. 278–299.

[29] C. Boura and A. Canteaut, "A zero-sum property for the Keccak-f permutation with 18 rounds," NIST mailing list, 2010. [Online]. Available: http://www-roc.inria.fr/secret/Anne.Canteaut/Publications/zero_sum.pdf

[30] H. Kaeslin, *Digital Integrated Circuit Design. From VLSI Architectures to CMOS Fabrication*. Cambridge, UK: Cambridge University Press, 2008.

[31] A. Juels, "RFID security and privacy: a research survey," *IEEE J. Select. Areas Commun.*, vol. 24, no. 2, pp. 381–394, Feb. 2006.

[32] Y. Eslami, A. Sheikholeslami, P. G. Gulak, S. Masui, and K. Mukaida, "An area-efficient universal cryptography processor for smart cards," *IEEE Trans. VLSI Syst.*, vol. 14, no. 1, pp. 43–56, Jan. 2006.

[33] M. O'Neill, "Low-cost SHA-1 hash function architecture for RFID tags," in *Proc. of the Workshop on RFID Security RFIDsec*, 2008.

[34] M. Feldhofer and J. Wolkerstorfer, "Strong crypto for RFID tags - a comparison of low-power hardware implementations," in *Proc. of the IEEE Int. Symposium on Circuits and Systems (ISCAS)*, New Orleans, LA, May 2007, pp. 1839–1842.

[35] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES implementation on a grain of sand," in *Proc. of IEE Information Security*, vol. 152, Oct. 2005, pp. 13–20.

[36] D. Hein, J. Wolkerstorfer, and N. Felber, "ECC is ready for RFID - a proof in silicon," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, vol. 5381. Springer Berlin / Heidelberg, 2009, pp. 401–413.

[37] L. Batina, J. Guajardo, B. Preneel, P. Tuyls, and I. Verbauwhede, "Public-key cryptography for RFID tags and applications," in *RFID Security*. Springer US, 2009, pp. 317–348.

[38] J. Wolkerstorfer, "Is elliptic-curve cryptography suitable to secure RFID tags?" in *Proc. of the Workshop on RFID and Lightweight Crypto*, Graz, Austria, 2005.

[39] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, ser. Lecture Notes in Computer Science, vol. 3156. Springer Berlin / Heidelberg, 2004, pp. 85–140.

Luca Henzen (S'08) received his M.S. degree in electrical engineering from the Swiss Federal Institute of Technology Zurich (ETHZ), Zurich, Switzerland, in 2007.

He then joined the Integrated Systems Laboratory of the ETHZ as Research Assistant, where he is currently pursuing the Ph.D. degree. His research interests include the design of VLSI circuits for cryptographic applications and low-power systems.

Jean-Philippe Aumasson got his M.S. degree in computer science at Paris VII university (France) in 2006, and his doctoral degree in computer science at the Swiss Federal Institute of Technology Lausanne (EPFL) in 2009.

He has been a doctoral researcher at the University of Applied Sciences Northwestern Switzerland in Windisch during his PhD. Since 2010, he is working as a cryptography engineer for Nagravision SA in Cheseaux, Switzerland. His research interests are analysis and design of symmetric cryptographic algorithms.

Dr. Aumasson is a member of the International Association for Cryptologic Research.

Willi Meier got his diploma in mathematics in 1972, and his doctoral degree in mathematics in 1975, both at the Swiss Federal Institute of Technology Zurich (ETHZ).

He has been a guest researcher at the universities of Oxford and Heidelberg and a research assistant at University Siegen (Germany). Since 1985 he is a professor of mathematics and computer science at the University of Applied Sciences Northwestern Switzerland in Windisch. His present interests are analysis and design of cryptographic primitives like stream ciphers and hash functions. He is an associate member of ECRYPT II, and an associate editor of the Journal of Cryptology.

Prof. Meier is a member of the International Association for Cryptologic Research.

Raphael C.-W. Phan (M'03) obtained his B.Eng. (Hons.) degree in computer engineering in 1999, and M.Eng.Sc. and Ph.D. degrees in cryptography in 2001 and 2005 respectively.

He is a lecturer in the Electronic & Electrical Engineering department of Loughborough University, UK. He researches in diverse areas of security and privacy, ranging from cryptology through side-channel attacks and smart cards to authentication protocols.

Dr. Phan has served in the technical program committees of IEEE conferences including ICC, Globecom, WCNC and PIMRC.