Hardware Evaluation of the Stream Cipher-based Hash Functions RadioGatún and irRUPT

L. Henzen, F. Carbognani, N. Felber, and W. Fichtner
Integrated Systems Laboratory, ETH Zurich, Switzerland
henzen@iis.ee.ethz.ch

Abstract

In the next years, new hash function candidates will replace the old MD5 and SHA-1 standards and the current SHA-2 family. The hash algorithms RadioGatún and irRUPT are potential successors based on a stream structure, which allows the achievement of high throughputs (particularly with long input messages) with minimal area occupation. In this paper, several hardware architectures of the two above mentioned hash algorithms have been investigated. The implementation on ASIC of RadioGatún with a word length of 64 bits shows a complexity of 46 k gate equivalents (GE) and reaches 5.7 Gbps throughput with a $3 \cdot 64$-bit input message. The same design approaches 120 Gbps on ASIC with long input messages (63.4 Gbps on a Virtex-4 FPGA with 2.9 kSlices). On the other hand, the irRUPT core turns out to be the most compact circuit (only 5.8 kGE on ASIC, and 370 Slices on FPGA) achieving 2.4 Gbps (with long input messages) on ASIC, and 1.1 Gbps on FPGA.

1 Introduction

Hash functions are cryptographic primitives that take an input message and generate an output referred to as hash value or digest. Since a message of arbitrary bit-length produces a fixed-length output, different messages might be hashed into the same digest. The fundamental characteristic of hash functions is, therefore, the ability to keep the probability of such a collision low. Moreover, for a given output it should be computationally infeasible to recover any of the generator inputs. These properties are referred to as *collision resistance* and *preimage resistance* [7]. Digital signature, message authentication, and data integrity make an extensive usage of hash functions.

Recently, the old hash primitives MD5 and SHA-1 have been broken (collisions found) [5, 16]. Hence, the National Institute of Standards and Technologies (NIST) have standardized the SHA-2 family [8]. Since these algorithms rely on similar computational structure and a successful collision attack on SHA-2 could have catastrophic effects for today's digital signature schemes, NIST introduced in 2007 a public call for new cryptographic hash algorithms [9]. The intent of the competition is to identify modern secure hash functions and to define the new SHA-3 family.

Although the security provided by an algorithm is the most important evaluation criterion, in a second phase of the competition the hardware profile of the candidates will be evaluated. In particular, NIST will emphasize the computational efficiency and the implementation costs of every algorithm. Therefore, candidate algorithms should be suitable for hardware implementations (ASIC or FPGA) in various embedded systems, where high-speed, low-power, or limited memory requirements may play a cardinal role.

The hash function RadioGatún [2] has been presented as a competitive alternative algorithm in terms of performances, while the cryptographic algorithm irRUPT is a specific hash function based on the symmetric cryptographic primitive EnRUPT [10]. Since RadioGatún and irRUPT have an intrinsic simplicity in computing the hash value, their hardware implementations are expected to be very compact, while reaching very high throughput. This work presents the first complete VLSI characterization of the RadioGatún and irRUPT hash functions, through the investigation of six hardware architectures. Their suitability for hardware implementation is eventually confirmed by the achieved results and the comparison with other hash algorithms.

The remainder of this paper is organized as follows. Section 2 introduces the computational processes of RadioGatún and irRUPT and the required parameters. Section 3 describes the methodologies used in the implementation of the hardware architectures. The achieved experimental results are presented in section 4 with a comparison between the analyzed algorithms and other hash functions. Eventually, section 5 draws the conclusions.

2 Algorithm Specification

Hash functions generally work over an iterative model, which computes the fixed-size hash value by processing successive fixed-size input blocks. A message of arbitrary

978-3-9810801-5-5/DATE09 © 2009 EDAA

length is divided into l-bit blocks (with a padding process on the last block), that are sequentially injected into an internal collision-resistant compression function. The compression function generates an n-bit intermediate digest using the input block and the previous intermediate result. The last intermediate digest is defined as the final n-bit hash value, after all input blocks are processed. This model is also called the Merkle-Damgård (MD) construction [7]. The security level of MD-based hash algorithms is defined by the collision resistance of the compression function. The SHA-1 and SHA-2 families are hash functions based on this iterative structure, as well as MD5 [13] and the NESSIE candidate Whirlpool [12].

However, several attacks on MD-based hash functions have recently been demonstrated, turning the MD construction into an insecure model. Alternative designs rely on modern stream ciphers [15]. This is the case of RadioGatún and irRUPT. Although the former relies on the alternating-input IMF (iterative mangling function) [2] and the latter on a stream mode of the cryptographic primitive EnRUPT [10], the resulting stream cipher-based hashing schemes present similarities. The peculiarity resides in the computational schedule of the algorithms. Instead of running a compression process for every l-bit input block, the whole message is first entirely inserted into the hash function and, then, it is elaborated to obtain the related hash value. The phase schedule of this structure (see Fig. 1) consists of a message input phase, where the message blocks are sequentially inserted, a blank round phase, i.e., a series of blank rounds, and the digest output phase, where every iteration generates the elements of the final hash value.

RadioGatún and irRUPT are defined over a simple round function, which updates the internal state variables at every round. In the input phase, the iterations of the round function are alternated with the insertion of the input blocks. The output phase executes the round function over the state and outputs some elements of the state variables as the hash value.

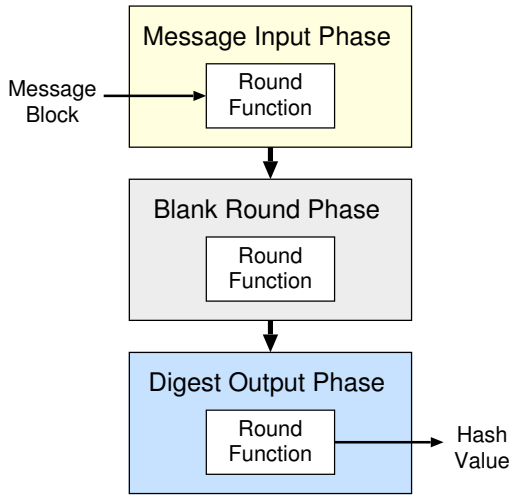

Figure 1. Phase schedule of the hash functions RadioGatún and irRUPT.

Algorithm 1 Alternating-input construction

$Z \leftarrow$ RadioGatún$w(M)$
$(A, B) = (0, 0)$
for $i = 0, 1, \ldots, n_m - 1$ **do**
  $(A', B') = F_i(A, B, M)$
  $(A, B) = \mathrm{R}(A', B')$
end for{*Injection*}
for $i = 0, 1, \ldots, n_b - 1$ **do**
  $(A, B) = \mathrm{R}(A, B)$
end for{*Mangling*}
for $i = 0, 1, \ldots, n_z - 1$ **do**
  $(A, B) = \mathrm{R}(A, B)$
  $(z_{2i}, z_{2i+1}) = (a_1, a_2)$
end for{*Extraction*}

2.1 The RadioGatún Hash Function

RadioGatún is based on the alternating-input construction of IMF. As described in Alg. 1, the phases of Fig. 1 are injection, mangling, and extraction. The l-bit input block $M = (m_i, m_{i+1}, m_{i+2})$, where every element is a w-bit word ($l = 3w$), is mapped into the state variables A (mill) and B (belt) by the function $F_i()$. After n_m iterations a message of $3wn_m$ bits is taken. Then, $n_b = 16$ blank rounds are repeated over the state A and B, followed by n_z iterations of the round function and the extraction of two words from A. The central component is the round function $R()$. Its belt-and-mill structure (see Alg. 2) consists of the *Mill()* and the *Belt()* functions with the *Mill to Belt* and the *Belt to Mill* operations. The mill A consists of 19 words a_i, while the belt B has 13 stages b_i of three words $b_{i,j}$.

The $Mill()$ function consists of four invertible transformation (see Alg. 3). Among these, the first is the only one introducing non-linearity in the hashing process. The input

Algorithm 2 Round function

$(A', B') \leftarrow \mathrm{R}(A, B)$
for $i = 0, 1, \ldots, 12$ **do**
  $b'(i) = b(i - 1 \bmod 13)$
end for{*Belt function*}
for $i = 0, 1, \ldots, 11$ **do**
  $b'(i + 1, i \bmod 3) = b'(i + 1, i \bmod 3) \oplus a(i + 1)$
end for{*Mill to Belt*}
$A' = \mathrm{Mill}(A)$
for $i = 0, 1, 2$ **do**
  $a'(i + 13) = a'(i + 13) \oplus b(12, i)$
end for{*Belt to Mill*}

Algorithm 3 *Mill* function (indices are modulo 19)

$A' \leftarrow \text{Mill}(A)$
for $i = 0, 1, \ldots, 18$ **do**
  $a'(i) = a(i) \oplus [a(i+1) \vee \overline{a(i+2)}]$
end for
for $i = 0, 1, \ldots, 18$ **do**
  $a(i) = a'(7i) \ggg i(i+1)/2$
end for
for $i = 0, 1, \ldots, 18$ **do**
  $a'(i) = a(i) \oplus a(i+1) \oplus a(i+4)$
end for
$a'(0) = a'(0) \oplus 1$

mapping function $F_i()$ is defined as

$$(A', B') = \begin{cases} a'_{i+16} = a_{i+16} \oplus m_i & i = 0, 1, 2 \\ a'_i = a_i & else \\ b'_{0,i} = b_{0,i} \oplus m_i & i = 0, 1, 2 \\ b'_i = b_i & else \end{cases} \quad (1)$$

The length of the final hash value is given by $n = 2wn_z$.

2.2 The irRUPT Stream Hashing Mode

irRUPT has been defined as a stream hash function. It is based on the symmetric cryptographic primitive EnRUPT. As stated by the author, EnRUPT can be used to construct fast and secure hash functions. Its simple structure uses only exclusive-ORs (XORs), word-wise rotations, and modulo 2^w additions. Multiplication by nine is turned into the addition of the word with its '$\ll 3$' part.

The basic computation is carried out in the $ir1(p)$ small round function (application of EnRUPT), defined as

$$\begin{array}{lcl} f & = & [(2x_{r-1} \oplus x_{r+1} \oplus d \oplus r) \ggg w/4] \cdot 9, \\ x_r & = & x_r \oplus f, \\ d & = & d \oplus f \oplus p \oplus x_{h/2+r}, \\ r & = & r + 1, \end{array} \quad (2)$$

where X is the $h = \frac{2n}{w}$-word state variable, d a w-bit feedback variable, r the round count, and p the input word. The global round function $ir2s(p)$ executes eight sequential calls of $ir1(p)$. Therefore, the security/performance trade-off of irRUPT is defined, by setting $s = 4$, as advised in [10].

The computations of the irRUPT hashing scheme are resumed in Alg. 4. The generation of the hash value starts with a process phase, where $ir2s(m_i)$ combines the input words of the message M with the state X ($l = w$). Then in the finalize phase, h rounds of $ir2s()$ are computed without inserting any input. At the end, the words of the final hash value are extracted from d after additional $h/2$ blank rounds of $ir2s()$.

Algorithm 4 stream hashing (indices of x are modulo h)

$Z \leftarrow$irRUPT$w(M)$
$X = 0, d = 0, r = 1$
for $i = 0, 1, \ldots, n_m - 1$ **do**
  $ir2s(m_i)$
end for{*Process*}
for $i = 0, 1, \ldots, h - 1$ **do**
  $ir2s(0)$
end for{*Finalize*}
for $i = 0, 1, \ldots, h/2 - 1$ **do**
  $ir2s(0)$
  $z_i = d$
end for{*Output*}

3 Hardware Implementations

The core of the implementation of the presented hash algorithms is the single round function ($R()$ or $ir2s()$) and the memory unit to store the intermediate state variables. Since RadioGatún and irRUPT do not use substitution boxes, no additional memories are required. A further irRUPT architecture with a reduced round function has been investigated to achieve different performance trade-offs. Every design is controlled by a dedicated control unit, which computes the round count and routes the signal inside the cores.

3.1 RadioGatún Architecture

RadioGatún enables different specifications of word size. The w parameter, besides defining the capacity of the security level, constitutes the smallest computational unit, on which the arithmetic of the function is defined. The SHA-2 family includes the SHA-256 algorithm, working with $w = 32$ and SHA-512 with $w = 64$ bits. Hence, the choice to implement RadioGatún32 and RadioGatún64 allows a more direct comparison between the algorithms. Furthermore, the extraction count n_z has been fixed to four, in order to generate a 256-bit output in RadioGatún32 and a 512-bit output in RadioGatún64.

The high-level architecture of the RadioGatún cores is depicted in Fig. 2. The single round block is connected with the register-based memory to store the state variables A and B every clock cycle. Two multiplexers select the inputs of the round function, depending on the current hashing phase. The round unit is based on the four operations, defined in Alg. 2. The functions *Mill()* and *Belt()* are computed in parallel, as well as the *Mill to Belt* and the *Belt to Mill* transformations. The *Belt()* function is a simple rotation of the words inside the belt variable B, while the two last transformations are carried out by XORs. The most complicated and area-expensive module is the *Mill()* function, which limits the maximum working frequency of the round block. The '$\lll k$' operator is a k-bit rotation towards

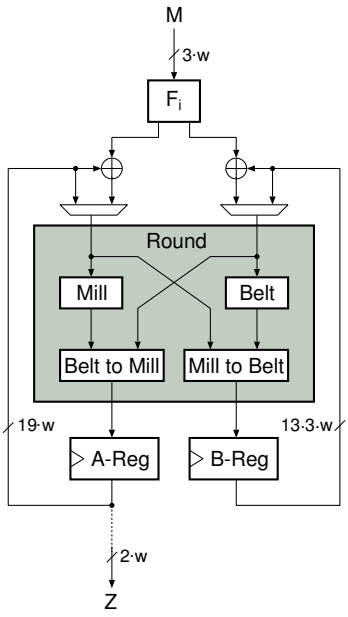

Figure 2. Block diagram of the implemented RadioGatún architecture.

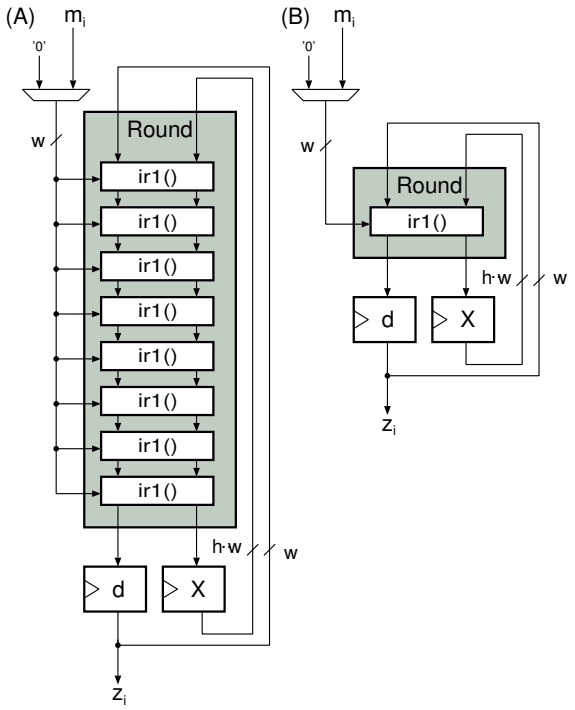

Figure 3. Block diagram of the implemented 8-irRUPT (A), and 1-irRUPT architecture (B).

less significant bits; in hardware, it is translated into a direct re-routing of the bits inside the word.

During the extraction phase, the circuit outputs two words of the final hash value every cycle.

3.2 irRUPT Architectures

The cryptographic primitive EnRUPT has been specified for a word length w of 32 or 64 bits. Both irRUPT32 and irRUPT64 have been implemented. They use a fixed $h = 16$, which means a generation of a 256-bit hash value by the former and a 512-bit hash value by the latter.

The flow dependencies between the eight calls of $ir1()$ inside the global round function $ir2s()$ inhibit every parallelization effort. In fact, a single $ir1()$ updates the word x_r, which is then used by the following $ir1()$ to update the next word. Two architectures were, therefore, devised. The first is the basic $8 \times ir1()$ design, isomorphic to the algorithm specification (see Fig. 3 (A)). Every cycle, a complete $ir2s()$ round is computed. Afterwards, the new X state and the intermediate digest d are stored inside the memory. In the output phase, d constitutes a word of the final hash value.

The second architecture is based on an iteratively decomposed $ir2s()$ function (see Fig. 3 (B)). It consists of the same memory for X and d with the addition of only one $ir1()$ module. Eight iterations (clock cycles) are, therefore, needed to execute the complete $ir2s()$ round. The input message words m_i keep constant for eight cycles at the input of the circuit. The final digest word z_i is generated only every eight cycles, during the output phase.

The design strategy used in the implementation of the $ir1()$ function is based on a progressive word-shift of the state (see Fig. 4). Instead of using time-expensive multiplexers and demultiplexers to route the elements of X, the $ir1()$ module takes the words x_o and x_2 as inputs, while updating the word x_1. The new intermediate digest d is

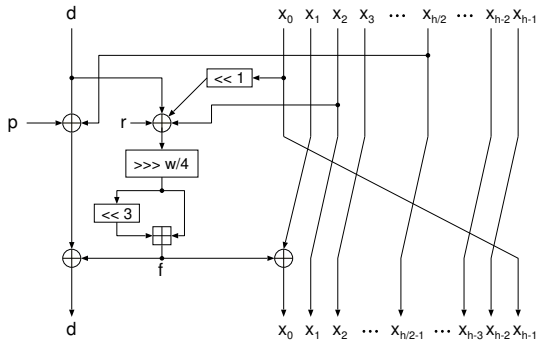

Figure 4. Word-shift design of the $ir1()$ function. All connections are w bits wide.

Table 1. ASIC post-synthesis implementation results.

Ref.	Function	Area [kGE]	Frequency [MHz]	Cycles	Throughput [Gbps]		HW-eff. [Kbps/GE]		Tech. [µm]
					Single	Large	Single	Large	
Ours	RadioGatún32	21.42	667	21	3.048	64.000	142.3	**2,987.9**	0.18
Ours	RadioGatún64	45.94	625	21	5.714	**120.000**	124.4	2,612.1	0.18
Ours	8-irRUPT32	15.63	105	40	1.342	3.354	85.8	260.8	0.18
Ours	8-irRUPT64	33.30	90	40	2.315	5.787	69.5	208.2	0.18
Ours	1-irRUPT32	**5.83**	592	320	0.947	2.367	162.3	**406.0**	0.18
Ours	1-irRUPT64	11.70	485	320	1.553	3.883	132.8	331.9	0.18
[6]	SHA-256	22.03	794	68	5.975		271.3		0.13
[14]	SHA-256	15.10	190	72	1.349		87.6		0.18
[6]	SHA-512	43.33	746	84	9.096		209.9		0.13
[14]	SHA-512	30.75	170	88	1.969		64.0		0.18
[14]	Whirlpool	167.37	187	10	9.588		57.3		0.18
[14]	Whirlpool	38.91	102	21	2.485		63.9		0.18
[14]	Whirlpool	52.79	262	21	6.382		120.9		0.18

computed, by using the same word $x_{h/2}$. To preserve the functionality of the algorithm, every word is shifted down by one r index after the computation of $ir1()$. More precisely, the $ir1()$ function constantly uses the word pair $\{x_0, x_2\}$ and $x_{h/2}$ respectively to compute f and to update d. The final word shift operation is equivalent to the increment of the index variable r by one.

4 Results and Performance Comparison

The specific structure of the two presented hash functions leads to performance results that strongly depend on the input message size. This is due to the different number of clock cycles that is needed in the input phase to acquire the whole message. The throughput is given by:

$$T = f\frac{S}{(n_m + n_{b+z})} = f\frac{S}{(\frac{S}{kw} + n_{b+z})}, \qquad (3)$$

where f is the circuit frequency and S the size of the padded message. The parameter n_{b+z} corresponds to the number of cycles needed by the blank and the output phase, while the constant k defines the number of words injected per cycle into the hashing core:

- RadioGatún: $n_{b+z} = n_b + n_z = 20$, $k = 3$;
- 8-irRUPT: $n_{b+z} = h + \frac{h}{2} = 24$, $k = 1$;
- 1-irRUPT: $n_{b+z} = 8(h + \frac{h}{2}) = 192$, $k = \frac{1}{8}$.

Fig. 5 shows the S-dependency of the circuit throughput normalized over the frequency. Using long message sizes, the speed converges to the $f \cdot kw$ value. Compared to hash functions based on the Merkle-Damgård construction, where the throughput remains constant regardless of the message length, RadioGatún and irRUPT take large advantage of the input dimension, in addition to the efficiency and the speed of their round functions.

The hashing cores have been coded in functional VHDL and synthesized with Synopsys Compiler, targeting a 0.18 µm CMOS technology. The hardware analysis is completed by the evaluation on a Xilinx Virtex-4 FPGA device. Tab. 1 and Tab. 2 list the resource utilization, the maximal clock frequency, and the throughput of the six RadioGatún and irRUPT architectures with additional designs of Whirlpool and the SHA-2 family. For the ASIC cores, the hardware-efficiency of the circuit has also been reported. For the sake of completeness, the throughputs are computed

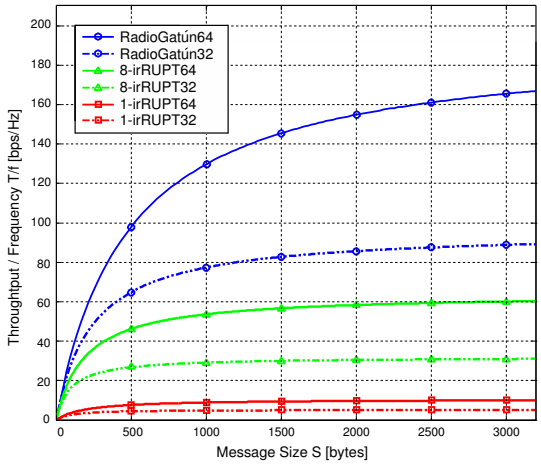

Figure 5. Throughput/message size trade-off of the implemented architectures.

Table 2. FPGA performance comparison of the hash functions.

Ref.	Function	Area [Slices-BRAM]	Frequency [MHz]	Throughput [Gbps]		FPGA Type
				Single	Large	
Ours	RadioGatún32	1,501-0	361	1.649	34.620	XC4VLX100
Ours	RadioGatún64	2,937-0	330	3.018	**63.387**	XC4VLX100
Ours	8-irRUPT32	948-0	49	0.632	1.581	XC4VLX100
Ours	8-irRUPT64	2,375-0	36	0.914	2.287	XC4VLX100
Ours	1-irRUPT32	**370-0**	286	0.457	1.149	XC4VLX100
Ours	1-irRUPT64	740-0	230	0.736	1.841	XC4VLX100
[4]	RadioGatún[64]	3,031-0	327	2.854		XC4VLX25
[11]	SHA-256	755-1	174	1.370		XC2PV
[11]	SHA-512	1,667-1	141	1.780		XC2PV
[3]	Whirlpool	2,118-32	220	5.380		XC4V
[1]	Whirlpool	**376-0**	214	0.082		XC2VP40

using two different message lengths. In the "single" case, RadioGatún and irRUPT hash respectively a single $3w$-bit block and a $16w$-bit message. The "large" entry in the tables reports the throughput in case of long messages (when the speed of RadioGatún and irRUPT approaches $f \cdot kw$).

The results point out a decisive superiority of RadioGatún in terms of speed. The stream structure and the high frequency of the RadioGatún architectures are the main asset, with respect to the other algorithms. On the other hand, the 1-irRUPT architecture represents the most compact circuit, suitable for resource constrained environments.

5 Conclusions

This paper presents the first complete VLSI characterization of the hash functions RadioGatún and irRUPT. The stream hashing structure is not only a plausible alternative to the Merkle-Damgård construction in terms of security, but it also shows a remarkable flexibility for hardware integration. The proposed RadioGatún family exhibits by far the highest throughput, compared with the SHA-2 family and the Whirlpool function, particularly in case of long input messages. Among the low-area designs, the 1-irRUPT circuit is the most compact architecture.

References

[1] T. Alho, P. Hamalainen, M. Hannikainen, and T. D. Hamalainen. Compact hardware design of whirlpool hashing core. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pages 1–6, Nice, Apr. 2007.

[2] G. Bertoni, J. Daemen, G. V. Assche, and M. Peeters. Radiogatun, a belt-and-mill hash function. In *Proc. of the Second Cryptographic Hash Workshop*, pages 24–25, Aug. 2006.

[3] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Merged computation for whirlpool hashing. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pages 272–275, Munich, Germany, Mar. 2008.

[4] P. Kitsos and B. Prasad. A system-on-chip design of the radiogatun hash function. In *Proc. of the Int. Conf. on High Performance Computing, Networking and Communication Systems*, July 2007.

[5] V. Klima. Finding MD5 collisions - a toy for a notebook. Cryptology ePrint Archive, Report 2005/075, 2005. http://eprint.iacr.org/.

[6] 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, 8th International Workshop, WISA*, pages 102–114. Springer, 2007.

[7] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

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

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

[10] S. O'Neil. EnRUPT first all-in-one symmetric cryptographic primitive. eSTREAM, ECRYPT Stream Cipher Project, (SASC 2008), 2008. http://www.enrupt.com/.

[11] L. S. Ricardo Chaves, Georgi Kuzmanov and S. Vassiliadis. Improving SHA-2 hardware implementations. In *Proc. of the Workshop on Cryptographic Hardware and Embedded Systems*, pages 298–310, Oct. 2006.

[12] V. Rijmen and P. Barreto. The whirlpool hash function, 2003. http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html.

[13] R. Rivest and S. Dusse. The MD5 message digest algorithm, Apr. 1992. Working Group Internet Draft, RFC 1321.

[14] A. Satoh. ASIC hardware implementations for 512-bit hash function whirlpool. In *Proc. of the Int. Conf. on Circuits and Systems*, pages 2917–2920, May 2008.

[15] M. Turan, O. Ozugur, and O. Kurt. hash function designs based on stream ciphers. In *Proc. of the Conf. ISC'07*, 2007.

[16] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.