2008 International Conference on Reconfigurable Computing and FPGAs

Celator: a multi-algorithm cryptographic co-processor

Daniele Fronte
Aix-Marseille Université, IM2NP
Atmel
Marseille, France
Email: daniele.fronte@atmel.com

Annie Perez
Aix-Marseille Université, IM2NP
Marseille, France
Email: perez@polytech.univ-mrs.fr

Eric Payrat
Atmel
Marseille, France
Email: eric.payrat@atmel.com

***Abstract*—Nowadays hi-tech secure products offer more services and more security. The corresponding market is now oriented towards more flexibility. As an answer we propose here a Multi-algorithm Cryptographic Co-processor called Celator. A main processor entrusts to the Celator the cryptographic tasks like encrypting or decrypting data blocks using secret key encryption algorithms such as Advanced Encryption Standard (AES) or Data Encryption Standard (DES). Moreover Celator allows hashing data using the Secure Hash Algorithms (SHA). These algorithms are frequently implemented in hi-tech secure products in software or in hardware mode. Celator corresponds to their flexible hardware implementation. Moreover an user can (under specific conditions) implement its own cryptographic algorithm in Celator.**

Celator can perform an AES encryption with a throughput of 47 Mbps, the DES encryption with a throughput of 26 Mbps, condense a 512 bit SHA message with a throughput of 36 Mbps. Finally we report performance comparisons among Celator, General Purpose Processors, and some dedicated and dynamically reconfigurable circuits.

I. Introduction

This paper presents a multi-algorithm cryptographic co-processor called Celator. Celator can be programmed in order to execute the Advanced Encryption Standard (AES) algorithms [1], the Data Encryption Standard (DES) [2] and the Secure Hash Algorithms-256 (SHA-256) [3]. Celator is conceived to be made of Standard Cell resources and integrated in an ASIC for embedded circuits.

Secure embedded FPGA can be used to reconfigure such a coprocessor [4]. However, Celator does not include any FPGA. The adopted solution for Celator is based on a 4x4 Processing Elements (PEs) systolic array [5], which seems a good solution to compute all matrix format data. A single PE can operate 1 byte word data. PE array's data path is programmable, just as the Finite State Machine (FSM) which controls the PE array is, too. Because of these two programmable elements, Celator can be programmed. Results show that Celator is a good trade off with respect to the execution cycles, areas and flexibility between a dedicated hardware macro and a General Purpose Processor (GPP). The Celator cryptographic systolic approach was first used for AES only [6], [7]. The Celator programmable systolic approach was first studied in [8].

The rest of the paper is organised as follows: section II introduces the AES, DES and SHA-256 algorithms and their mapping into a systolic PE array. In section III the Celator architecture is detailed, as well as its instruction set. Section IV shows how Celator performs AES, DES and SHA-256. Celator performances are presented in section V and compared to the performances of other processors or cryptographic machines. Finally some conclusions are given in section VI.

II. The Algorithms into the PE array

A. AES

The AES specifies a Federal Information Processing Standards [1] approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt or decrypt information. The AES algorithm is capable of using cryptographic keys of 128, 192 and 256 bits. These different versions are called AES-128, AES-192 and AES-256 respectively. In this paper we adopt the AES-128 as the AES algorithm. The plain text consists of 128-bit data blocks. Each block can be managed as a matrix of 4x4 bytes. In Celator data are computed by a 4x4 PE array and each PE works on 8-bit data. Each element of the matrix (Table Ia) can be mapped to the Processing Element device (Table Ib). As a consequence, each byte to be encrypted or to be decrypted is mapped to a single PE, and each AES transformation can be mapped into the 4x4 PE array.

TABLE I: Data Mapping to the PE array.

(a) ***The 4x4 byte matrix for an AES-128 data block.***

d_{11}	d_{12}	d_{13}	d_{14}
d_{21}	d_{22}	d_{23}	d_{24}
d_{31}	d_{32}	d_{33}	d_{34}
d_{41}	d_{42}	d_{43}	d_{44}

(b) ***The 4x4 8-bit PE array of Celator.***

PE_{11}	PE_{12}	PE_{13}	PE_{14}
PE_{21}	PE_{22}	PE_{23}	PE_{24}
PE_{31}	PE_{32}	PE_{33}	PE_{34}
PE_{41}	PE_{42}	PE_{43}	PE_{44}

The AES encryption process includes 10 rounds. Each round (excepting the first one and the last one) involves the following transformations: Sub-Bytes, Shift-Rows, Mix-Columns, Add-Round-Key. More information about the AES transformations are given in [1].

978-0-7695-3474-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ReConFig.2008.76

B. DES

DES [2] is a block cipher which operates on plaintext 64-bit blocks and returns ciphertext blocks of the same size.
The DES is made of an Initial Permutation IP, of 16 rounds of transformations and of a Final Permutation FP. In each round an "xor" operation and a "f" function. The "f" function includes the E expansion, the P permutation and 8 S substitutions.

Unlikely AES permutations, DES permutations are bitwise permutations and thus their implementation in Celator requires more time with respect to a byte permutation. Their execution by Celator, which works on 32-bit wide words, shows how Celator can also work with bit format data.

C. SHA-256

The SHA-1 and the SHA-2 (i.e. SHA-256, SHA-384, and SHA-512) are four secure hash algorithms specified by [3]. These algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a *message digest*.
Each algorithm can be described in two steps: *preprocessing* and *hash computation*. Preprocessing involves padding a message, parsing the padded message into m-bit blocks (m= 512 for SHA-1 and SHA-256, m= 1024 for SHA-384 and SHA-512), and setting initialization values to be used in the hash computation. The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest.

From [3], 8 32-bit intermediate variables (a, b, c, ..., g, h) are required by SHA-256. These variables are initialized by 8 32-bit constants given by the standard $(H_1, H_2, \ldots, H_8)$.

Therefore SHA-256 operates on 32-bit words. Since as a single PE can compute a 8-bit data, four PE can be concatenated in order to compute a 32-bit data.

III. Celator Hardware Architecture

This section details the Celator and the architecture of the whole computer system around the Celator. The whole computer system (Fig. 1) includes a main processor (an ARM-based CPU), a Main Memory, the IF unit and Celator. Celator includes a PE array, a Controller and the CRAM.

The main processor gives to the Celator the data to be encrypted, decrypted or condensed. It has to specify the algorithm to use. The CPU and Celator communicate through an interface allowing both of them to work at different frequencies.

A. The PE array

The PE array is the *data path* of Celator, and consists of a systolic array of 4x4 Processing Elements. All PE are identical. The PE array is controlled by the Controller (section III-B). Systolic array processors [5] can have mono-dimensional, two-dimensional or three-dimensional I/O connections. The 3D I/O

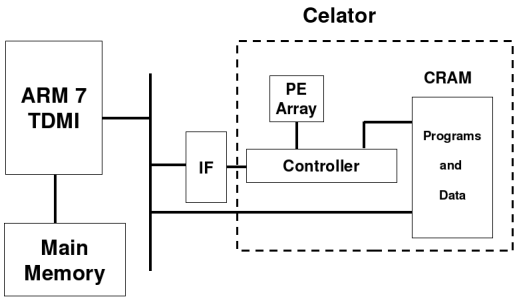

Fig. 1: Whole computer system architecture

connections need more surface than 1D I/O. For Celator we chose the 2D I/O, which seems to be a good trade off with respect to the connection and allows Celator to execute several algorithms.

Thus, by the 2D I/O, the whole PE array has (Figure 2):

- four 32-bit data inputs/outputs, i.e. one per cardinal direction;
- four multiplexers (MUX_N, MUX_E, MUX_W, MUX_S) which the input/output of the PE array allow to be switched among: North, East, West and South input/outputs. The multiplexers can also make PE array linked in a toric fashion, which can be useful to perform operations such as rotations around the PE array.

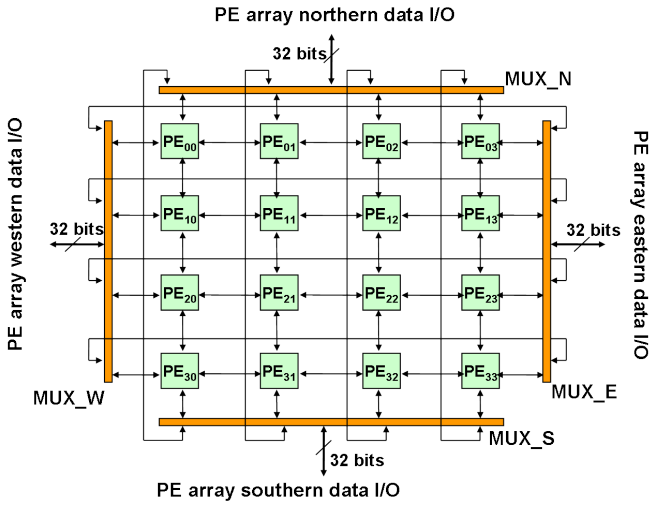

Fig. 2: PE array architecture

Each PE receives control signals from the Controller in order to compute the data. These control signals can:

- load data from northern, eastern, western, southern PE, the CRAM, the CPU, or an external source
- select the registering data between the computed data or the previous data
- select the PE data out between the previously registered data or the non registered data
- select one of the ALU operations

- read/write from western/northern PE and write/read into eastern/southern PE

All PE array multiplexers are configured by the Controller, and the configuration can be modified at each clock cycle. Therefore, the PE array data path is programmable.

Each Processing Element (PE) includes (Figure 3):

- one Arithmetic and Logic Unit (ALU);
- two 8-bit registers A and B;
- four 8-bit data inputs, i.e. one per cardinal direction;
- one 8-bit data output, which is linked to the data input of first near neighbor PEs;
- two multiplexers, which allow to switch the input/output of the PE array from: North, East, West and South;
- two multiplexers, which select the data input for registers A and B, between the input of the PE and the data output of the other register, respectively.

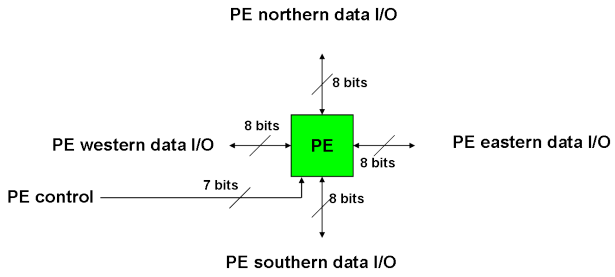

Fig. 3: The Processing Element data Input/Output

As for PE array, all PE multiplexers are configured by the Controller, and the configuration can be modified at each clock cycle. Therefore, the PE data path is programmable, just as the FSM and the PE array data path are.

In order to compute the data, each PE receives the control signals from the Controller. These control signals can:

- load data from northern, eastern, western or southern PE
- enable the register(s) for write mode
- select the input data of the registers A and B, between the ALU output and the other register output
- select one of the ALU operations
- select the PE output: north, east, west and south

The PE's ALU can perform the following algebraic and logic operations on 8-bit operands (for operands larger than 8 bits, more PE must be concatenated):

- `xtime`
- xor
- and
- not
- bypass
- adder modulo 256: over 32 bits, 4 PE are used, i.e. all PE of a row of the PE array
- right shift: 1 bit is right shifted between 2 (either vertical or horizontal) neighbor PE
- left shift: 1 bit is left shifted between 2 (either vertical or horizontal) neighbor PE

Each Processing Element can also load/store a 8-bit data from one of its nearest neighbour PE.

The `xtime` is a special AES dedicated operation, needed by the AES MixColumns transformation. This operation consists of a multiplication by hexadecimal "0x02", and can be implemented at byte level as a left shift and a subsequent conditional bitwise xor with "0x1B" (for more details about `xtime` function see [1]).

B. The Controller

The Controller consists of an FSM and several registers (Figure 4). The FSM loads the instructions in the CRAM, decodes and executes them by generating the control signals for the PE array, the CRAM, and for the IF. It also controls the PE array data inputs and outputs.

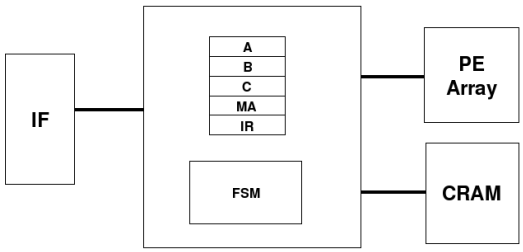

Fig. 4: The Controller architecture

Four types of FSM instructions have been defined:

- Type (1) instructions: RAM access
- Type (2) instructions: PE operations (the PE array is divided into 4 PE rows)
- Type (3) instructions: PE operations (the PE array is divided into 4 PE columns)
- Type (4) instructions: FSM computations

The five registers of the Controller are:

- A: 32-bit register used when look up tables are operated;
- B: 32 bit register;
- C: 12-bit counter register;
- MA: 12-bit Memory Address register;
- IR: 29-bit Instruction Register;

The Controller generates the control signals for the PE array and the instructions for the CPU. It also controls the PE array data inputs and outputs.

The FSM instructions are simple, they are not AES dedicated instructions. They are basic instructions which can be reused to implement other cryptographic algorithms. For instance, an FSM instruction can copy data from the CRAM to the PE array and vice-versa, with a specified offset or not, or can rotate data in the PE array. The FSM instructions are stored in the CRAM.

C. The CRAM

The Celator RAM, or CRAM, can store 32-bit data and instruction words (Figure 5). Data include the encrypting or decrypting input/output data, as well as all data needed by the encrypting/decrypting process, e.g. the Sbox for the AES. Programs stored in CRAM include the micro-instructions for the Controller. The CRAM is a double port RAM, therefore both CPU and Controller can access to the CRAM in read/write mode.

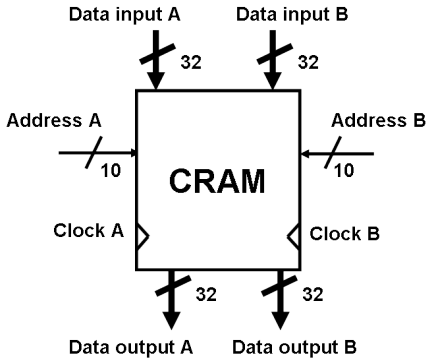

Fig. 5: The CRAM architecture. It is a double port RAM.

IV. Algorithms implemented using the PE array

Celator can perform all 4 AES transformations that are required by AES encryption, decryption and key scheduling. The Controller can execute a whole AES by combining these transformations.

The PE array works like a systolic Complex Input Multiple Data (CIMD) systolic Processor [5]. The CRAM contains the instructions for the FSM. These instructions are fetched by the FSM and then are executed by the PE array. The CPU first loads the cryptographic algorithm, e.g. the AES, into the CRAM and then the CPU starts Celator. The CPU controls Celator via the IF unit and can modify some instructions of the cryptographic algorithm instruction sequence or select a different algorithm, too. Therefore, the CPU can ***program*** Celator.

A. The AES transformations

Sub-Bytes: the Sbox table is stored in the CRAM. The Controller FSM provides signals to make a look-up table of each data byte stored in each PE. An FSM register is also used.

Shift-Rows: each row of the PE array is left shifted by 1, 2, or 3 positions respectively for row 1, 2 or 3.

Mix-Columns: the Mix-Columns fundamental operation is the multiplication $S'(x) = A(x) * S(x)$. Each byte of the state $S(x)$ is multiplied by the polynomial vector $A(x)$, as described in [1], by the $xtime$ operation and then $xored$ par columns. Both $xtime$ and xor operations are provided by the ALU.

Add-Round-Key: the key values are loaded from the CRAM into the PE array and then xored with the array data.

B. The DES transformations

All DES permutations can be performed in the following way. The 64-bit input word $A = a_1 \dots a_{64}$ must be multiplied by an identity matrix. As a_i are binary elements, the multiplication results will be 64 words made of one "1" and 63 "0". Then these words must be rotated and xored each other.

Since a single PE compute 8-bit data, four PE can be concatenated in order to compute 32-bit data. In this way, the PE array can be considered as four 32-bit PE rows. In order to execute DES, 2 PE rows are used to operate a whole 64-bit data block.

The DES S substitutions are executed like the AES SubBytes transformations.

C. The SHA-256 transformations

We assume that the preprocessing of the input data is performed by the CPU, and Celator will compute the hash computation only. Therefore here we will detail the implementation of the SHA-256 hash computation only.

From [3], the required variables are the following ones:

- 8 32-bit intermediate variables (a, b, c, ..., g, h);
- 48 32-bit W_j variables;
- The following 8 32-bit variables ($ROTR$ is a circular right rotation): T_1, T_2, Ch, Maj, Σ_0, Σ_1, σ_0, σ_1;
- The result is $H_N = (H_{n1}, H_{n2}, \dots, H_{n8})$, i.e. on 8 32-bit data;

Four PE are concatenated to operate on 32-bit words.

D. ECB and CBC modes

Celator is able to encrypt and decrypt an AES data block both in Electronic Codebook (ECB) mode and in Cipher Block Chaining (CBC) mode [9]. Fig. 6 show the difference between encrypting a gray scale image ([10], Fig. 7a) in ECB (Fig. 7b) and CBC (Fig. 7c) modes.

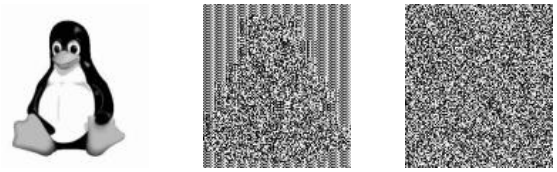

Fig. 6: AES encrypting in ECB and CBC modes

V. Results and discussion

Celator performances are given thereafter and compared with the performances of other cryptographic structures. We chose to compare Celator with GPPs [11], [12], [13], dedicated hardwired macros [14], [15], [16] and also with a dynamically reconfigurable architecture [17].

The whole computer system presented in Fig. 1, were developed in Verilog at the Register Transfer Level, simulated by Mentor ModelSim and synthesized by Synopsys DC with Atmel Standard Cells resources and the 130 nm technology. These same tools were also used to determine the ASIC areas

of Atmel and of AVR microprocessors. Results are shown and commented below.

The version of Celator presented here needs 514 clock cycles and performs 47 Mbps to encrypt an AES 128-bit data block in ECB mode. In CBC mode an AES 128-bit data block is encrypted in 524 clock cycles and performed at 46 Mbps (Table II). Even though a dedicated hardware macro is faster than our solution (the AES hardware macro of Atmel takes 40 cycles, i.e. a cycle for each AES transformation), it needs a larger area. GPPs such as the ARM 7 TDMI and ARM 9 processors are larger and slower than Celator. The GPP AVR has the same size but it is slower than Celator.

TABLE II: Dedicated macro, GPP and reconfigurable macro areas and throughput (AES CBC mode) comparison

	cycles	**Area** (mm^2)	**Bit-rate** (Mbps)	**Tech.** (nm)
Helion [18]	–	0.1	650.0	130
Atmel [14]	40	0.2	480.0	130
Celator	524	0.1	46.0	130
μ **PiCoGa** [17]	285	11.0	90.0	90
μ **AVR** [11]	5000	0.1	1.0	130
μ **ARM 7** [12]	3600	0.4	2.3	130
μ **ARM 9** [19]	830	1.1	31.0	130

Macros presented in [20], [21], [22] have throughput performances (bits per cycle) less than or equal to the Atmel Macro [14], i.e. 3.2 bits per cycle for encrypting an AES 128-bit data block in ECB mode. Dedicated hardware macros (Tables II) have a higher bit-rate (bit per second) than Celator but unlike Celator they are not programmable. A dynamically reconfigurable architecture such as PiCoGa [17] achieves a higher bit-rate than Celator but, even if PiCoGa is reconfigurable, it is far larger than Celator.

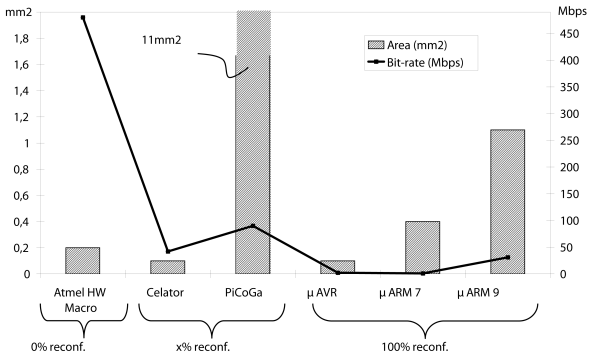

Fig. 7: Areas and Throughputs for AES-128 in CBC mode encrypting of a special purpose macro, of multi-algorithms processors and of general purpose processors.

Since Celator is programmable, we do not need to modify its architecture in order to encrypt a DES data block.

Celator needs 476 clock cycles to encrypt a DES 64-bit data block. It achieves 26 Mbps throughput when operated at 190 MHz. This throughput is smaller than other circuits' performances. The slowest operation in DES Celator implementation is the permutation. In DES there are up to 5 permutations, and 3 of them are executed 16 times. A PE array like Celator seem to not be fully suitable to execute these permutations, that are bitwise operations, and leave the most of the PEs unused.

The software solution proposed in [16] operates on a 64-bit processor, which is very suitable to manage 64-bit data block. S-boxes are hardwired therefore this solutions is not fully programmable, unlike Celator is.

The Atmel hardware macro needs 16 clock cycles to perform a DES encryption, i.e. a cycle per DES round. This is one of the best throughput suitable for embedded systems. Again, this solution is not programmable, unlike Celator is.

TABLE III: Comparisons of areas and performances for encrypting a DES 64-bit data block in EBC mode.

	Type	**Cycles**	**Freq.** (MHz)	**Bit-rate** (Mbps)
Atmel	HW	16	100	400
Saqib [15]	FPGA	–	–	274
Ebiham 1 [16]	SW	140	300	167
Ebiham 2 [16]	SW	417	300	46
Celator	HW/SW	476	190	26

The FPGA based solution has a high throughput, thanks especially to its parallelized S-boxes.

Celator here needs 2720 clock cycles to hash a SHA 512-bit data block. It achieves 36 Mbps throughput when operated at 190 MHz. This throughput is smaller than other circuits' performances. The slowest operations in SHA Celator implementation are the data transfers among all registers that store the SHA values (SHA values are 8 32-bit registers). In order to speed up the data transfers, thus the SHA execution, some local memory blocks must tu be included in the Celator implementation (other than the CRAM).

TABLE IV: Comparisons of areas and performances for hashing 512-bit data block, in according with SHA-256.

	Type	**Cycles**	**Freq.** (MHz)	**Bit-rate** (Mbps)
Rchaves [23]	HW	65	–	1400
Iahmad [24]	HW	–	–	1000
Cadence [25]	HW	70	133	971
Celator	HW/SW	2800	190	36

To conclude this result section, we can say that Celator has a good percentage of programmability given by the portfolio of its instructions, which can modify the PE array datapath and select up to eight ALU operations. It also results

clear that the Celator represents a good trade off among dedicated hardware macros (which are not programmable), the dynamically reconfigurable architectures (which have a good percentage of programmability) and GPPs (which are fully programmable) in terms of areas, and throughput and flexibility (Fig. 7).

VI. Conclusions

The multi-algorithm Crypto-Co-Processor called Celator presented in this paper can perform the AES, DES and SHA algorithm. It is made of 16 Processing Elements (PEs) arranged in a systolic fashion (PE array), of a local RAM (the CRAM) and of a Controller, which includes an FSM. The FSM reads the instructions in the CRAM, and the PE array executes them under the control of the FSM. Celator can be programmed by the CPU. Given its structure based on a PE array, all matrix format data can be easily computed. Celator is a good trade off choice of crypto-processor with respect to the number of gates, the number of execution cycles and the programmability. Celator performances are situated among those of dedicated hardware macros and those of GPPs.

In this paper the security aspects are not analysed. We consider "writing" or "reading" operations into/from a register or the CRAM as trusted operations. Given the structure of Celator, during the encrypting or decrypting operations, data can be masked; for instance data can be xored once or several times with random data in order to become more difficult to be intercepted by hackers through Side Channel Attacks [26], [27].

Next steps of our investigations will be the implementation of other algorithms in Celator. PE array architecture will be modified, but its surface will not be considerably affected, thanks to its generic structure and its generic instruction set. The long term goal will be the study and development of a trusted platform to secure exchanges of data between the CPU and Celator, as well as between the CPU and an external data source.

References

[1] "Specification for the advanced encryption standard (AES)," Federal Information Processing Standards Publication 197, 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] National Institute of Standards and Technology, "Data encryption standard (DES)," Federal Information Processing Standard 46-3, Tech. Rep., 1999. [Online]. Available: http://csrc.nist.gov/publications/fips/fips46

[3] N. I. of Standards and Technology, "Secure hash standard (SHA-2) (+ change notice to include SHA-224)," Federal Information Processing Standards Publication 180-2, Standard, Aug. 2002. [Online]. Available: http://csrc.nist.gov/publications/

[4] N. Valette, L. Torres, G. Sassatelli, and F. Bancel, "Securing embedded programmable gate arrays in secure circuits," *ipdps*, vol. 0, p. 226, 2006.

[5] P. Frison, E. Gautrin, D. Lavenier, and J. L. Scharbarg, "Réseaux systoliques spécifiques à base du processeur," Institut National de Recherche en Informatique et en Automatique (INRIA), France, Tech. Rep., 1990. [Online]. Available: http://www.inria.fr/

[6] D. Fronte, A. Perez, and E. Payrat, "System and method for encrypting data," U.S. Patent US2 008 062 803, 2008. [Online]. Available: http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US2008062803&F=0

[7] ——, "The AES in a systolic fashion: Implementation and results of celator processor," in *IEEE International Conference on Electronics, Circuits, and Systems*, 2008.

[8] ——, "A multi-algorithm cryptographic co-processor," in *Computing, Communications and Control Technologies, in the context of the International Multi-Conference on Engineering and Technological Innovation*, 2008.

[9] N. Ferguson and B. Schneier, *Practical Cryptography*, C. A. Long, Ed. Wiley, 2003.

[10] Tux the penguin. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/a/af/Tux.png

[11] Atmel, "AVR processor 8 bits," Tech. Rep., 2006. [Online]. Available: http://www.atmel.com/products/AVR/

[12] (2007) Arm 7 tdmi. [Online]. Available: http://www.arm.com/products/CPUs/ARM7TDMI.html

[13] (2007) Arm 9 embedded core. [Online]. Available: http://www.arm.com/products/CPUs/families/ARM9Family.html

[14] Atmel Secure Terminals, "AT91SO100," Atmel, Tech. Rep., 2007. [Online]. Available: http://www.atmel.com/dyn/products/product_card.asp?part_id=3810

[15] N. A. Saqib, F. Rodríguez-Henriquez, and A. Díaz-Pérez, "A compact and efficient fpga implementation of the DES algorithm," 2004.

[16] E. Biham, "A fast mew DES implementation in software," 1997. [Online]. Available: http://www.cs.technion.ac.il/~biham/

[17] D. Thull and R. Sannino, "Performance considerations for an embedded implementation of oma drm 2," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 46–51.

[18] Standard AES cores, "Helion standard AES," Helion, Tech. Rep., 2007. [Online]. Available: http://heliontech.com/aes_std.htm

[19] C. Mucci, L. Vanzolini, F. Campi, and M. Toma, "Interactive presentation: Implementation of AES/rijndael on a dynamically reconfigurable architecture," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*. San Jose, CA, USA: EDA Consortium, 2007, pp. 355–360.

[20] S. Mangard, M. Aigner, and S. Dominikus, "Highly regular and scalable AES hardware architecture," *IEEE Transactions on Computers*, 2003. [Online]. Available: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/10557/33406/01581%498.pdf?arnumber=1581498

[21] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact rijndael hardware architecture with s-box optimization," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2001, pp. 239–254. [Online]. Available: http://www.springerlink.com/index/BC7DVD7YMADU3J8L.pdf

[22] S. Sharma and S. B. Sudarshan, "Design of an efficient architecture for advanced encryption standard algorithm using systolic structures," in *International Conference of High Performance Computing (HiPC)*, 2005. [Online]. Available: http://www.hipc.org/hipc2005/posters/systolic.pdf

[23] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Improving SHA-2 hardware implementations," in *CHES*, ser. Lecture Notes in Computer Science, L. Goubin and M. Matsui, Eds., vol. 4249. Springer, 2006, pp. 298–310.

[24] I. Ahmad and A. S. Das, "Hardware implementation analysis of SHA-256 and SHA-512 algorithms on fpgas," *Computers & Electrical Engineering*, vol. 31, no. 6, pp. 345–360, 2005.

[25] "Hashing algorithm generator SHA-256, cadence datasheet."

[26] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," in *The Journal of Cryptography*, Counterpane, University of Berkley and University of Princeton, 2000. [Online]. Available: http://www.schneier.com/paper-side-channel.html

[27] T. Lash, "A study of power analysis and the AES (recommendation for designing power analysis resistant attack)," in *MS Scholarly Paper*, George Mason University, 2002.