Energy Characterization of a Security Module in ARM Processor

Felipe dos Anjos Lima, Edward David Moreno, Dellano Oliveira D. dos Santos, and Wanderson Roger Azevedo Dias
(Corresponding author: Edward David Moreno)

DCOMP/UFS - Department of Computer Science, Federal University of Sergipe
Aracaju, Sergipe, Brazil
(E-mail:{felipes1474, edwdavid, dellano.lelo, wradias}@gmail.com)
(Received Apr. 26, 2013; revised and accepted Nov. 26, 2013)

Abstract

This article shows the results obtained during simulations that measured runtime and energy consumption of a security module (SEMO) when it executes in ARM processor. For the simulations, we considered the impacts of four algorithms (i.e. RSA, SHA-1, Random Numbers Generator and AES). We have used the Sim-Panalyzer simulator and obtained an average energy consumption (x2 Joules) and runtime (x26000 cycles). We also show the impact of some compiler optimizations in the energy consumption of the AES algorithm.

Keywords: AES, ARM, energy, RSA, security

1 Introduction

In the Knowledge era, information is increasingly spreading around the world. This is very substantial, owing to the fact that people are able to easily access the information they want. It only takes a few clicks on the internet, and everything is within reach. Therefore, this easiness brings some problems which need to be carefully analyzed, for instance, we know many people could act maliciously and try to gather information from others without proper permission.

By the time internet was created, people started to use it to perform several tasks, which cost long to be accomplished. For example, a bank transfer can be properly done with just a few clicks, without even leaving home. However, in order to accomplish a safe transaction, some password protection methods and information security mechanisms are needed, so that the site is reliable. If some computer hacker were able to intercept the users passwords, the damages to the user account could be irreparable. Furthermore, it is fundamentally important that the access, to the kind of system discussed above, is classified. For this purpose, some security mechanisms like password usage may be utilized. However, due to security reasons, these passwords may not be generated without any technique [5]., they must be created by reliable software so that they cannot be decrypted by unauthorized people.

Thus, security is a fundamental requirement at any serious computational system. Therefore, by the last years, several algorithms that aim to mask password keys, texts or any kind of confidential information were developed. Also, several security modules still being created trying to guarantee reliability at transactions performed by electronic devices, and the safety of the information stored. These modules can be implemented in hardware or software. In case they were implemented in hardware, they are expected to be embedded on chips, performing several security mechanisms.

The implementation of security modules in software follows the same philosophy, and also presents software components that protect data. Thus, the major objective of a security model is to provide a supervised system to execute its tasks independently, without having to deal with security issues. Therefore, in this article, we present a software implementation of a security module, called SEMO which is similar to one TPM (Trusted Platform Module) [3], which proved to be promising on the information security field. Then, we present analysis of our security module, making usage of the architectural simulation tool Sim-Panalyzer [1], in that we have obtained performance values by measuring runtime and energy consumption from our SEMO.

This article is divided in six sections. In Section 2 we present the SEMO module and its components; in Section 3 we depict one software implementation of the SEMO (TPM in software); Section 4 shows the simulations and analysis of the module; Section 5 presents the impact of the key seize on the energy consumption of AES algorithm, and finally, in Section 6 we present conclusions and ideas for future works.

2 SEMO - A Security Module

Our security module (SEMO) is similar to a TPM since it is a security module composed by a set of components with the intention of protecting the information stored on the device in that a TPM is coupled. This module can be developed on both hardware and software. In case it is implemented directly in hardware is known as TPM (Trusted Platform Module).

Every key generated by a cryptography process is encapsulated in the TPM so that it is not possible to access it from external models. Thus, external attacks aiming to discover passwords or stealing information may not succeed. In addition, the security module can also guarantee safety on both web browsers and email boxes. Figure 1 shows an example of a TPM. As we can observe on Figure 1, a TPM module holds a set of components which allows the execution of a set of security tasks. Its main units are described as follows:

- **I/O**: manages the information flow on the bus, directing messages to the appropriate components;
- **Cryptographic Processor**: executes cryptographic asymmetric operation and hashing utilizing well-known algorithms by the security community, like RSA, AES or SHA-1;
- **Key Generation**: creates pair of keys for asymmetric algorithms and symmetric keys;
- **Random Number Generation**: it is the source of randomness of a TPM. The TPM uses these random values on key generation and randomness on signatures;
- **SHA-1**: it is the hash function which is primarily used by a TPM, because he is a trustful implementation of a hash algorithm;
- **Opt-In**: provide protection mechanisms that allow the TPM to be turned on/off, or enabled/disabled;
- **Execution Engine**: execute programs according to the commands received by the TPM, using the values from I/O;
- **Non-Volatile Memory**: utilized for storing persistent identity and the state associated to the TPM;
- **Platform Configuration Register**: it is a local storage with 160 bits for measuring the discrete identity.

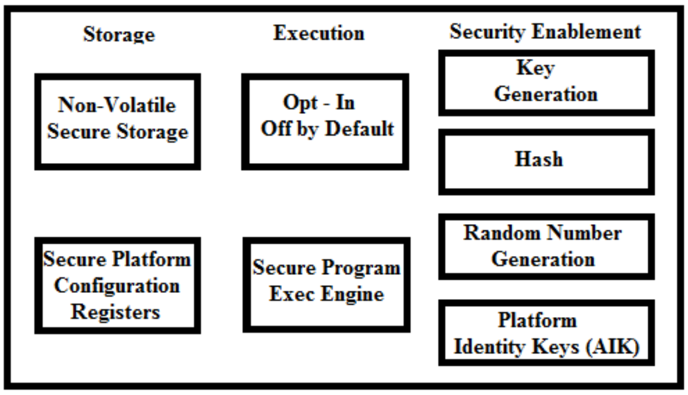
Storage
Execution
Security Enablement
Non-Volatile Secure Storage
Opt - In Off by Default
Key Generation
Hash
Secure Platform Configuration Registers
Secure Program Exec Engine
Random Number Generation
Platform Identity Keys (AIK)

Figure 1: Components of a TPM

3 Software Implementation of the SEMO

In this section, we describe our contribution, in other words, a TPM implementation in software, with some security components: namely, AES Engine, SHA-1 Engine, Random Number Generator and RSA Engine. Every component was implemented using C as its programming language. Figure 2 depicts the architecture from the SEMO module we have developed.

3.1 AES Engine

The AES Engine (AES - Advanced Encryption Standard) component that was implemented in our SEMO provides data encryption utilizing 128, 192 and 256-bit keys. The AES algorithm performs, during the encryption process, some operations over the data blocks received as input. Next, we shall present the operations performed by this component.

- **SubByte**: the bytes of the state variable are replaced utiizing a substitution table (S-BOX);
- **ShiftRow**: the bytes from each line of the state variable are rotated;
- **MixColumn**: each column from the state variable is transformed in another column through a modular multiplication;
- **AddRoundKey**: the key of the round is added to the state variable using XOR operation.

The four operations above can be inverted. In this manner, in order to perform the decryption of an encrypted text, the operation SubBytes, ShiftRow, MixColumn and AddRoundKey need to be inverted. The operations InvSubBytes, InvShiftRow, InvMixColumn and InvAddRoundKey operate over the encrypted data block.

3.2 SHA-1 Engine

The SHA-1 (Secure Hash Algorithm) Engine component increments an important document authentication function to the TPM, due to the fact that keys generated by the hash function from this algorithm are unique, in other words, two distinct documents do not own the same

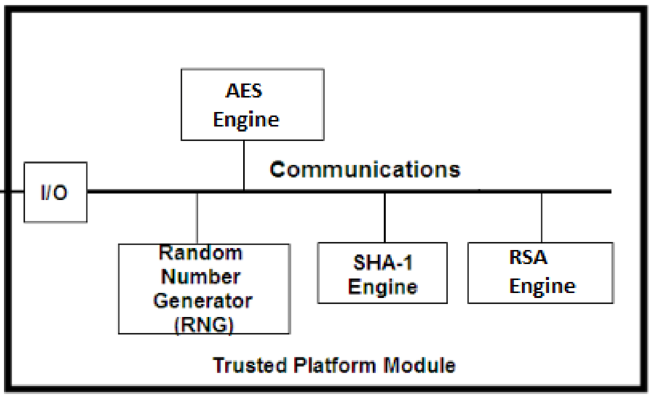

Figure 2: Components of our security module SEMO

representation. Therefore, during some transaction that demands a digital signature from the document, any modification on it would be perceived by the SEMO, and the authentication shall not be done.

3.3 Random Number Generator

The Random Number Generator (RNG) we implemented generates random values that are utilized by the RSA Engine Component for the encryption processes of data [5]. These values remain encapsulated inside the SEMO. However, the algorithm executed by the RSA Engine Component uses a random choice of prime numbers as a stop of the encryption process.

3.4 RSA Engine

The RSA (Random Scheduling Algorithm) Engine Component runs an encryption algorithm of public key, in that, every participant in the data transfer holds a public and a private key. These keys are pairs of integer numbers.

With the RSA Engine Component embedded on the architecture of the SEMO, our module inherited a new component that helps in the security process of information.

4 Simulation and Analysis of SEMO

In this section we present the simulation tool Sim-Panalyzer. We also present the analysis and simulations and their results, considering the impacts on energy consumption and performance at each one of the SEMO components.

4.1 Sim-Panalizer Simulator

Sim-Panalyzer is an energy consumption simulation tool, based on the SimpleScalar which simulates processors [2].

Table 1: Architectural components of Sim-Panalyzer

Architectural Component	Description
aio	Address bar of the input/output units
dio	Data bar of the input/output units
irf	Integer registers
fprf	Floating point registers
*il*1	Level 1 instruction caches
*dl*1	Level 1 data caches
*il*2	Level 2 instruction caches
*dl*2	Level 2 data caches
itlb	TLB instruction table
dtlb	TLB data table
btb	Branch Target Buffer
bimod	Switch Predictor
ras	Address stack result
logic	Random logic circuit
clock	System clock generator
uarch	Microarchitecture (the way USA is implemented in a processor)
fpu	Floating point unit
mult	Multiplication unit
alu	Arithmetic logic unit

SimpleScalar [1] simulates the computational architecture of a CPU platform, cache and memory hierarchy, and based on this model, it manages to simulate real programs over the specified platform. The processor utilized in this article comes from the ARM (Advanced RISC Machine) family [1].

Sim-Panalyzer was built based on the ISA (Instruction Set Architecture) from the families ARM [7] and Alpha of processors, obtaining great results on this kind of simulations [6]. Sim-Panalyzer has several components that altogether generate the total of energy consumption for the architecture [4]. Each one of these components plays an important role on measuring the total consumption of energy spent by the algorithms. Table 1 shows the components of Sim-Panalyzer that consume energy. The parameters for generating those vital parts of the computer were set as input for the Sim-Panalyzer simulator, which altogether generate the patterns of measurement of performance and energy consumption.

4.2 Analysis

While the simulations were performed on the Sim-Panalyzer tool, we have observed that SEMO obtained an average value of energy consumption of 2 Joules when a 1Kb File was subject to an encryption process. On Figure 3, we present the energy consumption of the components AES Engine, SHA-1 Engine, RNG and RSA Engine from our SEMO implementation, and then we detail the architectural elements which obtained the highest energy consumption values. On Figure 4 we present the runtime values per cycles from each one of these components.

As we can observe on Figure 4, the Component AES Engine presented the lowest consumption, this happens because the number of operations executed by the AES algorithm in this component is smaller than in the other ones. Also, the AES was designed to be utilized by embedded systems, which work with a lesser energy consumption. The RSA engine was the one that presented the largest energy consumption values in our SEMO. The components AES and RSA utilized 128-bit keys.

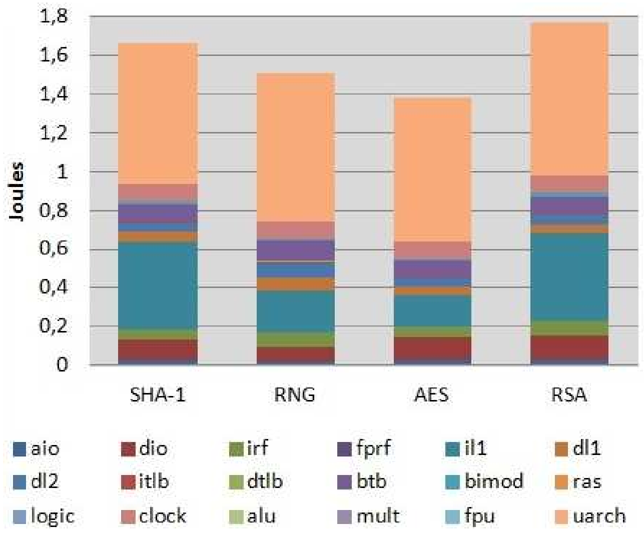

Figure 3: Energy consumption from SEMO components

Regarding the architectural components that are present on the ARM platform we utilized in the simulations, it is possible to perceive that the architectural elements uarch (microarchitecture - the way that ISA is implemented on a processor) and ill (instruction cache level L1) have obtained the highest energy consumption values.

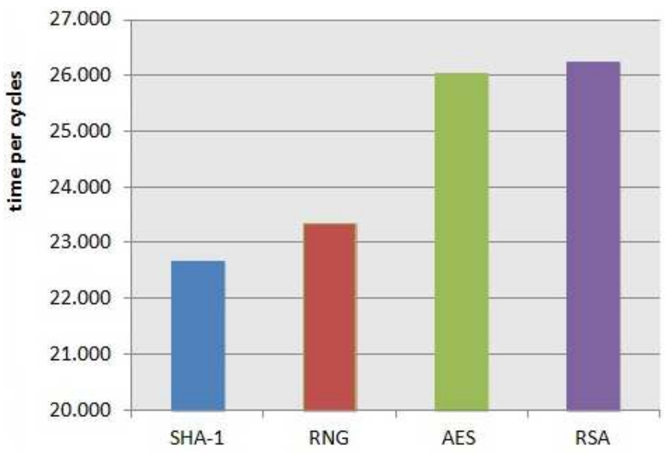

Figure 4: Performance of SEMO components

On Figure 5, we can observe that the components SHA-1 Engine and RNG presented very similar performance measurements, as well as AES Engine and RSA Engine. The simulations were performed in a netbook, with an Athon94 1.66GHz processor, and 2GB of RAM, running a Linux Ubuntu v11.04 operational system.

The SEMO can be implemented on several devices whether they are large or mobile ones. Considering the increasing miniaturization of computer devices, it is important that the existing software could run rapidly, without exhausting the energy available, because is not always possible to recharge batteries. Considering that a device that implements a SEMO could utilized all of its components to perform just one task, we present on Figure 5 the total energy consumption from our module.

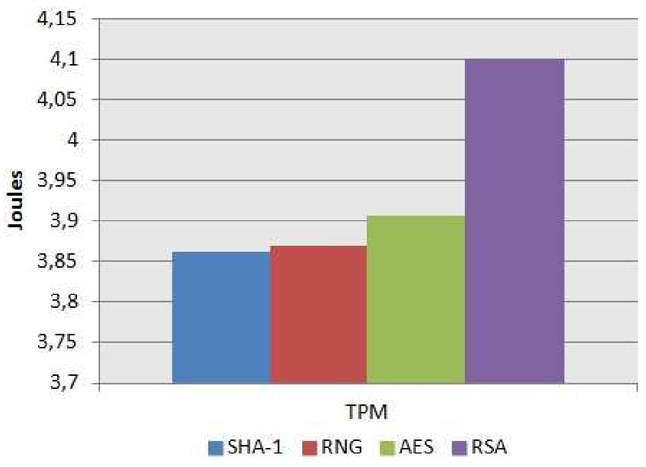

Figure 5: Energy consumption from the SEMO components

5 Detailed Analysis of the AES Component

In this section we make an analysis of the AES encryption algorithm, showing the impact the length of the keys provokes on the algorithms energy consumption and runtime.

5.1 Compiler Optimization

The executable codes generated by the optimizations are semantically equivalent to the original ones. In brief, they behave the same as the original codes given the same input [7]. With the increase in the utilization of embedded systems, the demand for code optimization has been stimulated so that energy consumption can be reduced [8]. In this article, we used gcc version 4.4.5 applying the following optimizations: -O0, -O1, -O2, -O3 and -Ip when compiling MiBenchs. The effects of each type of optimization are described hereafter:

- **-O0**: represents the default compilation, in other words, without any optimizations;

- **-O1**: this type of optimization enable some specific functions, for instance: (i) - Elimination of common subexpressions; (ii) - Global registry allocation;
- **-O2**: these are own specific functions, namely: (i) freorder-blocks: this function records blocks to be compiled in order so that algorithm processing costs can be reduced; (ii) falign-functions: this function reduces the information loading of compiling informations;
- **-O3**: these types of optimization also have specific functions, for example: (i) frename-registers: this function utilizes the maximum of all registers in a minimized way, thus avoiding false dependencies in the code; (ii) finline-functions: this function heuristically decides the simpler functions in order to declare them as static;
- **-Ip**: enables some optimizations like, for instance, dead-code elimination.

5.2 Key Size Impact on Energy Consumption

During the simulations we have utilized a 1Mb text file, which was encrypted and decrypted. We have also used 128, 192 and 256-bit keys (see Figure 6).

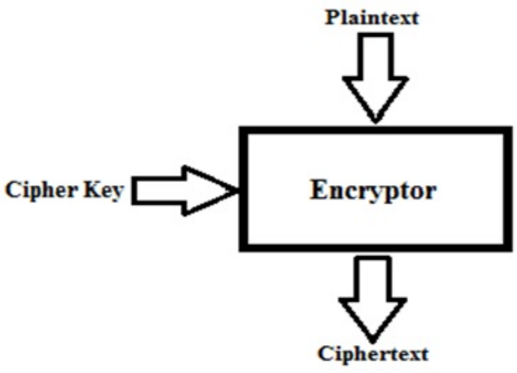

Figure 6: AES input and output

The file used as input for the simulations is handled in blocks by the algorithm. At each step, a 128-bit block is encrypted. The quantity of necessary operations to encrypt and decrypt each block depends on the length of the key used by the process. Table 2 presents the number of rounds executed by the AES algorithm for each cryptography key.

In this section, we present the impact key length has, highlighting the consumption of all architectural components of an ARM processor. As we can see on Figure 7, almost every architectural component obtained similar energy consumption value, therefore we highlight the dio component, which presented an average reduction of 23% when the key length increases. The simulation performed on Sim-Panalyzer showed that the O2 optimization presented a better performance when compared to other optimizations. The average gain was 10% for 128-bit keys when compared to other key lengths. Furthermore, it is possible to observe that the optimization flag Ip could not provoke any significant reduction on runtime, presenting similar results to the O0 which does not enable any greater impact optimizations (see Figure 8). Compiler optimizations are processes of improving the output of a compiler, which is an executable file, according to [7]. By means of switches, substitutions and even exclusion of structures, the compiler can return a faster program executable, consuming less memory.

Table 2: Number of rounds for encrypting/decrypting

Key Length	Number of Rounds
128 bits	10
192 bits	12
256 bits	14

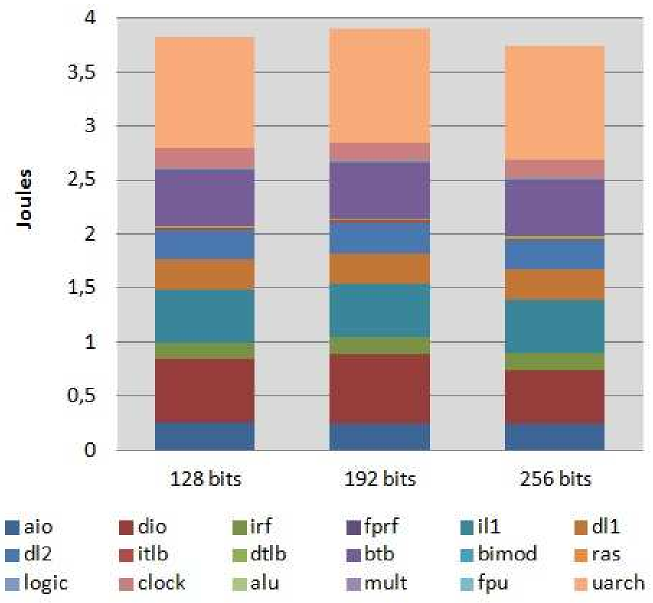

Figure 7: Characterization of energy consumption of AES

5.3 GCC Compiler Optimization Impacts on Energy Consumption

The analysis of the results show us that if the AES code is compiled utilizing O3 optimization, the average total consumption of all components is lower, when compared to the other optimization flags (see Figure 9).

The main optimizations enabled by the O3 flag are:

- **Inline-functions**: replaces every function call by the body of the function itself.

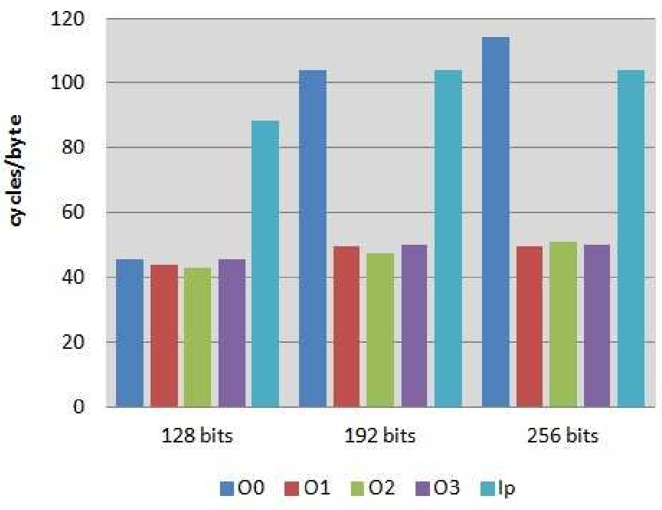

Figure 8: AES runtime measurements

- **Rename-register**: eliminates the dependencies between registers by reorganizing the stored values.

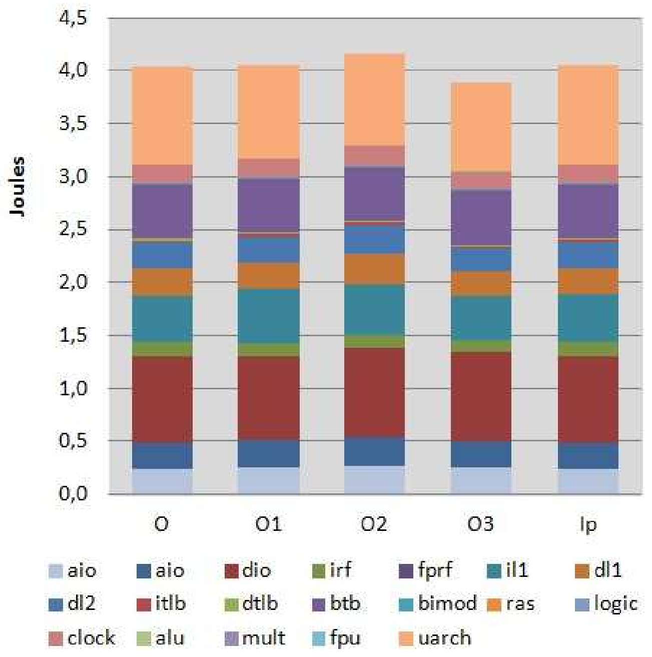

Figure 9: AES energy consumption

By activating the inline-functions optimization during the compilation step, every function present on the main program is replaced by the code that appears in the function body In this manner, the overhead on function call can be highly reduced. This explains why the energy consumption on AES is reduced, because even though the algorithm is compounded by just four functions, they a called at least 40 to 56 times, depending on the key used on encryption or decryption of data.

6 Conclusions and Future Works

In this work, firstly, we present the characteristics and functionalities of the SEMO (SEcurity MOdule). Secondly, we show its software implementation. And finally we present the simulation and analysis of this module, using the Sim-Panalyzer, where we concluded that our module had an average consumption of 2 joules and runtime of approximately 26 thousands cycles.

In addition, we have done a detailed analysis of the AES component, which is an algorithm people have given much attention to in the latest years. Several simulations were performed which had very satisfactory results, due to the fact that they could evidence the impact that key length caused on runtime and energy consumption of each component. Those results we obtained are now able to be used as a baseline for the implementation of security devices that demand less energy consumption.

As a future work, we plan to add more security components to the SEMO, as well as using it in a cellphone emulator. We also plan to develop a hardware implementation of the SEMO, and embedded it in a micro-controller and design an IP core in VHDL/Verilog and so that it can be used in several devices as notebooks, smartphones, ATMs and others.

References

[1] "The simplescalar-arm power modeling project,", June 2012. http://web.eecs.umich.edu/ panalyzer/.

[2] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH Computer Architecture News*, vol. 25, pp. 13–25, June 1997.

[3] Trusted Computing Group, June 2012.

[4] E. D. Moreno, F. M. R. Junior, F. A. Lima, and W. R. A. Dias, "Computer architecture under energy consumption vision," *International Journal of Computer Architecture Education*, vol. 2, pp. 5–8, Dec. 2013.

[5] E. D. Moreno, F. D. Pereira, and R. B. Chiaramonte, *Criptografia em software e hardware*. Novatec: São Paulo, Brazil, 2005.

[6] F. D. Pereira, E. D. M. Ordonez, and R. B. Chiaramonte, "Vliw cryptoprocessor: Architecture and performance in FPGAs," *International Journal of Computer Science and Network Security*, vol. 6, pp. 151–160, Aug. 2006.

[7] D. Seal, *ARM architecture reference manual*. Addison-Wesley Professional, 2nd edition edition, 2001.

[8] J. S. Seng and D. M. Tullsen, "The effect of compiler optimizations on pentium 4 power consumption," in *Proceedings of the Seventh Workshop on Interaction between Compilers and Computer Architectures*, p. 51, Sep. 2003.

Felipe dos Anjos Lima received his B. S. in Computer Science from Universidade Federal de Sergipe (UFS) in 2013. His areas of research interest include embedded systems, distributed computing and information security.

Edward David Moreno received the M.Sc. and Ph.D. degrees in Electrical Engineering from USP (University of So Paulo), SP, Brazil, in 1994 and 1998. During 1996 and 1997 he stayed as invited researcher at University of Toronto, Canada, and Chalmers University of Technology, Sweden. He is teacher at the UFS (Federal University of Sergipe), Aracaju, Sergipe, Brazil. Moreno has participated on 100 events as International Program Committee and he is editorial board of 4 important Journals: JUCS - Journal Universal on Computer Science, Springer Transactions on Computational Science and IJCSNS International Journal of Computer Science and Network Security and JCP - Journal of Computers. He has published five books about digital systems, FPGAs, Microcontrollers, Reconfigurable Computing and Hardware Security. The research areas are: computer architecture, reconfigurable computing, embedded systems, hardware security, power aware computing and performance evaluation.

Dellano Oliveira D. Santos received his B. S. in Computer Science from Universidade Federal de Sergipe (UFS) in 2013. His areas of research interest include embedded systems, distributed computing and information security.

Wanderson Roger Azevedo Dias received the B.S. in Computer Information Systems (2004), specialist in Software Development for Web (2007), M.Sc. and Ph.D. degrees in Computer Science by UFAM (Federal University of Amazonas), Manaus, Amazonas, Brazil, in 2009 and 2013. He is teacher at the IFS (Federal Institute of Sergipe), Aracaju, Sergipe, Brazil. The researches of teacher Roger are in the areas of: computer architecture, embedded systems, code compression, simulation architecture, hardware security, power aware computing, performance evaluation and FPGAs.