# Reconfigurable Security Architecture for Embedded Systems

*Abstract*— **Embedded systems present significant security challenges due to their limited resources and power constraints. We propose a novel security architecture for embedded systems (SANES) that leverages the capabilities of reconfigurable hardware to provide efficient and flexible architectural support to both security standards and a range of attacks. This paper shows the efficiency of reconfigurable architecture to implement security primitives within embedded systems. We also propose the use of hardware monitors to detect and defend against attacks. The SANES architecture is based on three main ideas: 1) reconfigurable security primitives, 2) reconfigurable hardware monitors and 3) a hierarchy of security controllers at the primitive, system and executive level. Results are presented for a reconfigurable AES security primitive within the IPSec standard and highlight the interest of such a solution.**

## I. Introduction

Security within embedded systems is becoming a major challenge since this condition is mandatory to enable the vision of ubiquitous computing. Two issues have to be considered when dealing with security; the first one is related to security primitives and protocols that are used to guarantee privacy and integrity of data, these security methods are mainly defined through standards. The second issue is related to attacks, as malicious users or funding organizations aim to defeat the security methods. Current solutions to address both issues are facing several gaps as demonstrated by Ravi et al [1].

First, from a performance point of view, the *processing*, the *battery* and the *flexibility gaps* have to be considered. The *processing gap* highlights that current embedded system architectures are not capable of keeping up with the computational demands of security processing. The *battery gap* emphasizes that the current energy consumption overheads of supporting security on battery-constrained embedded systems are very high. The *flexibility gap* shows that an embedded system is often required to execute multiple and diverse security protocols and standards. Second, from an attack point of view, the *tamper resistance* and the *assurance gaps* have to be addressed. The *tamper resistance gap* emphasizes that secure embedded systems are facing an increasing number of attacks from physical to software attacks, and the *assurance gap* is related to reliability and stresses the fact that secure systems must continue to operate reliably despite attacks.

Designing an embedded system architecture dealing with all these requirements is a challenging task. New solutions have to be defined in order to mitigate the costs of security. Two complementary approaches are considered in this paper that leverage the security within embedded systems. The first one is based on reconfigurable architectures that provide many interesting features to be selected as an efficient solution [2]. The second one is related to hardware monitoring to build Intrusion Detection Systems (IDSs). Indeed, software solutions show their limits when used in embedded systems as they are based on extensive audit of data, in form of system logging, which may require too much time and energy [3].

Thus, in this paper we propose a new approach to build embedded systems that takes benefit of both reconfigurable architectures and hardware monitors to increase security by detecting abnormal behaviors and by reacting appropriately. Such a solution enables the system to face an unsecured and evolving environment while meeting performance and constraints issues.

The remainder of this paper is organized as follows. Section 2 reviews previous efforts to build secure embedded systems. Section 3 presents our solution and shows how the security is enforced and how attacks can be fended off. In section 4 we deal with the AES security primitive within the IPSec standard to illustrate our concepts and to demonstrate their efficiency. Finally, section 5 concludes the paper and draws some perspectives.

## II. Related work

Existing efforts to promote security within embedded systems are mainly dealing with processor-based approaches [4][5][6]. These solutions are based on cryptography mechanisms to guarantee integrity and privacy of data and applications. Such solutions are very interesting, however as demonstrated by Ravi et al. [1] it is mandatory to define new alternatives to processor-based approaches as the costs of security using such solutions are very high. Other solutions can be considered using programmable hardware accelerators in order to mitigate the workload of processors. In [7] and [8] the authors propose cryptography processor or co-processor which can perform various execution modes and achieve high throughput. However, they do not address the attack issue and the energy efficiency metric is not considered. In [9] the authors focus on architecture support for energy-efficient security. In their work they deal with security primitives and security protocols but they do not consider the attack issue.

Another alternative is to consider reconfigurable architectures to implement security primitives instead of using programmable hardware accelerators. Several works have been published using such a solution [10][11][12] that have demonstrated its very high efficiency but none have focused on the mechanisms required to manage the flexibility of these primitives and to detect attacks.

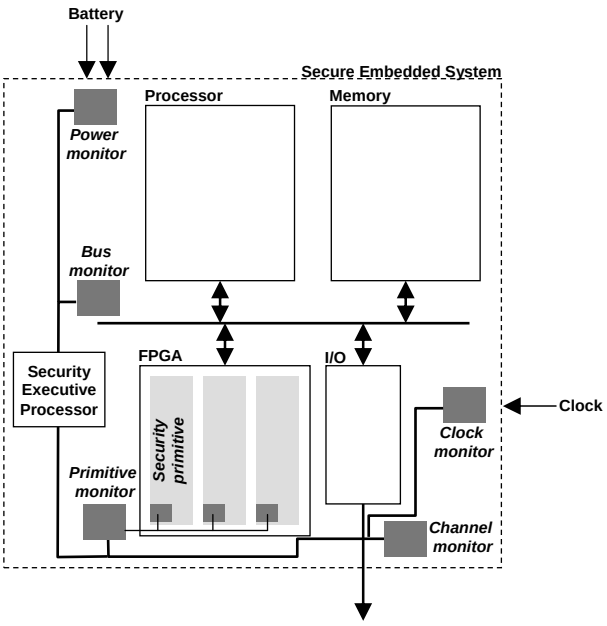


Fig. 1. The Security Architecture for Embedded Systems. The reconfigurable architecture contains the security primitives and the monitors protect the system

The concept of hardware monitoring has already been used for processor power reduction [13][14] and recently for power-attack [15]. In [15], the authors define the energy monitoring unit (EMU) which performs energy measurements to be compared to a set of reference energy signatures to detect when the system is under attack.

The work presented in this paper differs from these efforts in several respects. First, the underlying concept of our approach is to dynamically adapt the security protections in order to deal with dynamic constraints (i.e. attacks, performance, power). We propose an architecture that promotes the design of secure embedded systems by targeting all of the challenges stated by Ravi et al. [1]. Our approach allows the definition of a solution that leverages both flexibility and security within embedded systems. The performance and energy issues are considered by using reconfigurable security primitives which enable the system to provide several tradeoffs depending on the requirements and the security policy. The reliability issue is managed through the use of different implementations from low to high reliability (e.g. fault detection or fault tolerance). Second, we propose a hierarchy of hardware monitors in order to track the activity of the system. In our approach monitors provide different levels of flexibility which enables an evaluation of the right compromise between accuracy and simplicity which is mandatory to meet embedded system constraints.

## III. SANES: Security Architecture for Embedded Systems

### A. The SANES architecture: an overview

Our approach to protect embedded systems is by providing an architectural support for the prevention, detection and remediation of attacks. Most embedded systems are implemented as system-on-a-chip devices, where all important system components (processor, memory, I/O) are implemented on a single chip. We propose to extend the functionality of such systems to include both reconfigurable hardware and a continuous monitoring system that guarantees secure operations. Through monitoring, abnormal behavior of the system can be detected and hardware defense mechanisms can be employed to fend off attacks. Such an approach presents several advantages since application verification and protection is provided in dedicated hardware and not directly inside the application. The security mechanisms can be updated dynamically depending on the application running on the system which guarantees the durability of the architecture. Furthermore our approach focuses on embedded security and exploits the characteristics of embedded computations.

Figure 1 presents an overview of the architecture. As we can see several monitors are considered and track specific data of the system. The number and the complexity of the monitors are important parameters as they are directly related to the overhead cost of the security architecture. The role of these monitors is to detect attacks against the system. To provide such a solution, the normal activity (i.e. correct or expected) of the modules are characterized to detect irregular behaviors. Autonomy and adaptability have been stressed to build an efficient security-network of monitors. The monitors are autonomous in order to build fault tolerant systems; if one monitor is attacked the others can still manage the security of the system. The monitors are distributed to be able to analyze the different parts of the system (e.g. battery, buses, security primitives, communication channel).

Different levels of reaction are considered depending on the type of attack, reflex or global. Reflex reaction is performed by a single monitor; the response time is very short since no communication between the different monitors is required. Global reaction is performed when an attack involves a large modification of the system, in that case the monitors need to define a new global configuration of the system which leads to a longer response time. The monitors are linked by an on-chip intelligence network. This network is controlled by the Security Executive Processor (SEP) that acts as a secure gateway to the outside world. The SEP provides a software layer to map new monitoring and verification algorithms to monitors. In response to abnormal behavior, the SEP can issue commands to control the operation of the system. For example, it can override the power management or disable I/O operations.

### B. The reconfigurable architecture

The reconfigurable architecture within the system enables the implementation of security primitives. A security primitive corresponds to an agile hardware accelerator and performs a security algorithm (e.g. cryptography, IP filtering, key management). A device generally embeds several security primitives that work independently. Main goals of these modules are:

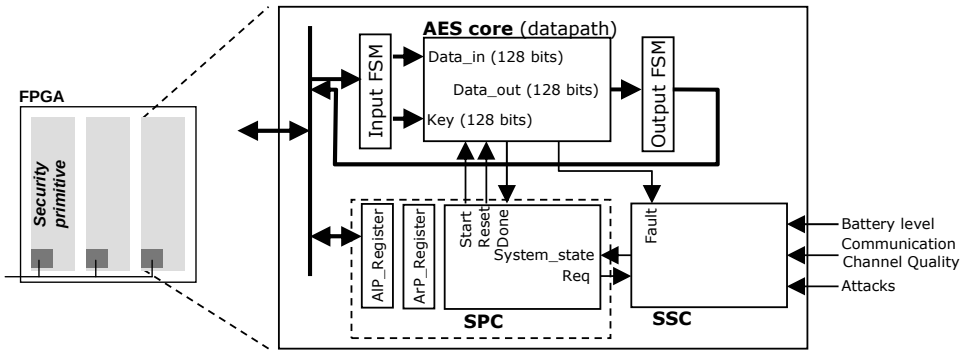


Fig. 2. The security primitive architecture. The Security Primitive Controller manages the flexibility of the primitive and the Security System controller deals with the detection of abnormal activity using specific sensors

- To speedup the computation of the security algorithm compared to software execution;
- To provide flexibility compared to a fixed implementation to be able to update the primitive or to switch from one primitive to another;
- To provide various tradeoffs in terms of throughput, area, latency, reliability, power and energy in order to meet real time constraints.

Figure 2 presents the security primitive architecture for an 128-bit AES algorithm. Three key components are considered, 1) the security primitive datapath, 2) the Security Primitive Controller (SPC), and 3) the System Security Controller (SSC) which is a monitor. An SPC is connected to the datapath in order to manage its flexibility. The SPC control tasks are related to reconfiguration of the datapath to change or adapt its architecture. The SPC is connected to the system processor in order to define the configuration of the security primitive. For example, in the case of cryptography it corresponds to the parameters of the algorithm (i.e. key size, mode and key value). A System Security Controller (SSC) is also connected to each security primitive to monitor the primitive and to check the system state to detect if some faults injection or abnormal operations are performed. The role of the SSC is to detect attacks against the primitive. The SSC is connected to other monitors to analyze the different parts of the system (e.g. battery, buses, other security primitives, communication channel).

### C. Detailed architecture of the primitive

The reconfigurable security primitive is composed of the datapath and the two previous controllers (SPC and SSC) as shown in figure 2. The SPC is connected to the system processor through a memory mapped mechanism (i.e. hardware accelerator). Depending on the primitive, different configuration registers are used to define its configuration. These registers provide the algorithm (i.e. execution mode and key size for cryptography algorithm) and architecture parameters (i.e. throughput, area and reliability). As stated before, the SPC manages the flexibility of the primitive. When the processor needs a security primitive it first configures the SPC which starts to check what execution modes can be used. Figure 3 presents the FSM corresponding to the SPC.

During the *Initialization* state, the SPC polls via the SSC the state of the system (i.e. battery level and communication channel quality) in order to define what implementations can be performed within the primitive. Once the algorithm and architecture parameters are checked, the SPC provides this information to the processor. During the *Configuration* state, once the processor has selected the algorithm parameters so that the security primitive can be configured, the SPC selects the corresponding configuration data (it corresponds to a bitstream) and starts the configuration of the datapath. On the *Run* state the security primitive is ready to run and to handle the data. While the datapath is running, the SPC regularly checks the system state through the SSC to define if the primitive needs to be reconfigured. Once the data has been computed, the security primitive can be stopped or can be removed from the reconfigurable hardware (*Stop* state). If the security primitive remains within the reconfigurable hardware this state corresponds to an idle state before running again the primitive. Finally, the *Security* state is particular in the sense it is always active. The *Security* state is driven by the SSC to indicate that a reconfiguration must be done in order to fend off or to anticipate an attack against the primitive. Whatever the state of the SPC, the *Security* state enforces the activation of the *Configuration* state to reconfigure the security primitive with the appropriate parameters.

### D. Dynamic security within the system: monitoring

Two main scenarios are considered in our work to protect the system from being pirated and to guarantee the execution of the security protections. The first one is managed by the SSC and deals with attacks (it relies on the security policy) and the second one by the SPC and deals with the flexibility of the primitive (it relies on the performance policy).

In the first scenario, the SSC can interrupt the SPC if an irregular activity is detected within the module or the system. In that case the SSC indicates to the SPC what configuration has to be implemented. Examples of attacks are: hijacking, denial-of-service (e.g. draining of battery or causing battery to overheat) and extraction of secret information (e.g. user's

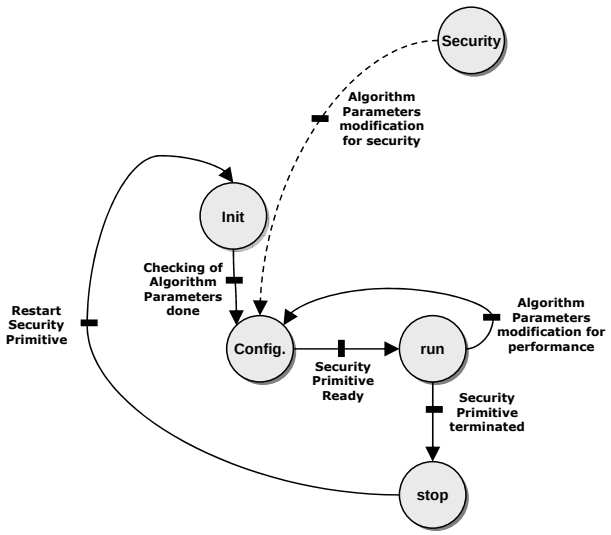


Fig. 3. The Security Primitive Controller FSM deals with the different states of the primitive to dynamically adapt the architecture of the primitive

phone book). In case of an hijacking attack the security primitive needs to be reconfigured with a safe configuration. In case of a denial-of-service attack the primitive needs to be enhanced by fault tolerance mechanisms to be able to guarantee its functionality and in case of an extraction of secret information attack, I/O of the primitive needs to be stalled.

One essential question is how an attack can be identified during run-time. It is not even theoretically possible to identify all attacks for any given system and thus we need to use an heuristic approach to address this problem. Our proposed monitoring system uses an approach where the current system behavior is compared to *normal behavior*. The expected behavior (namely *normal behavior*) of the system components is derived from off-line profiling runs in a secure environment. The aim is to detect behavior that deviates from a normal operation and thus detect an attack before the symptoms of the attack are evident. This capability allows the detection of intrusion, denial-of-service (e.g. drained battery as presented in [15]), hijacking, and even the attempt to extract secret information. For example, as will be detailed in the result section, by monitoring the data bus it is possible to detect that memory locations containing encryption keys are accessed even though no cryptographic operation is performed.

Profiling system behavior is a generally challenging problem. Today's computers run such a diverse and dynamically changing workload that any approach to characterize baseline system behavior is inconceivable. In our case we target embedded systems which simplifies the task. The embedded systems domain differs conceptually in two aspects that enables the definition of realistic solutions. These are:

- Simplicity of Workload. Many embedded systems are typically only executing a handful of programs at any given time. In most cases it is known beforehand which programs are run on the system and careful analysis can derive a baseline behavioral profile.
- Repetitiveness of Workload. Unlike workstation computer, where users can install and execute a large number of different programs, embedded systems are characterized by a less diverse, more *repetitive* workload. Many embedded systems perform simple control tasks (e.g. cell phone communicating with tower to determine if call is arriving), which by nature repeat the same instructions over and over again. Even though users switch between different modes of operation (e.g. between voice and messaging on a cell phone), the operations within each mode are often highly repetitive.

This simplicity and repetitiveness of embedded applications ensures that application profiling can indeed capture a large fraction of the application behavior and use this information for comparison to attack scenarios. Furthermore, it is important to define what has to be monitored. For embedded systems we believe that *power*, *clock*, *bus*, *security primitive* and *communication channel* monitors track most of the data within the system and enable the detection of main attacks.

### *E. Performance and security policies*

Once an attack has been fended off the SPC defines a new configuration to provide the best performance tradeoff (performance policy), for example in term of throughput versus energy when dealing with cryptography. Protected modes like fault tolerant architecture consume more area and power so it is essential to run these modes only when required and not by default to guarantee the power efficiency of the system. The security is also provided through the SPC since it continuously checks the state of the system to guaranty the best performance for the security primitive. Embedded systems are characterized by two main parameters, the power limitation and the evolving environment which leads to various level of quality of the communication channels. Hence, depending on both the SPC selects which parameters have to be considered. For example, in case of a best effort performance policy, when the level of battery is low or the channel quality decreases under some thresholds then the SPC reconfigures the module with a lower throughput but a better energy-efficient architecture. In case of guaranteed throughput, the SPC keeps the same parameters event if the thresholds are crossed.

The performance and security policies are essential issues to take benefit of the reconfigurability of the system and to provide efficient solutions. These policies are very dependent of the primitives and have to cope with their intrinsic specificities. The definition of these policies is beyond the scope of this paper, however designers must pay a particular attention to that point.

## IV. Security primitive and monitors: the AES case study

To demonstrate the concepts presented in this paper we have defined an agile security primitive and two hardware monitors. Our case study deals with the AES algorithm [16] since this FIPS standard has been selected by the National Institute of

| AES version | Slices (% of the total amount) | Slices (% compared to FB) | Period (ns) | Frequency (MHz) | Power (mW) | Power (% compared to FB) | Energy (nJ) | Throughput (Mbits/s) | Throughput (% compared to FB) | Energy efficiency (Gbits/J) |
|---|---|---|---|---|---|---|---|---|---|---|
| FB | 2192 (16%) | - | 26.4 | 37.8 | 996 | - | 316 | 403.7 | - | 0.4 |
| FB_FD | 2240 (16%) | +2.1 | 25.3 | 39.4 | 970 | -2.7 | 295 | 420.9 | +4 | 0.4 |
| FB_FT | 6302 (46%) | +65.2 | 25.2 | 39.6 | 1673 | +40.5 | 507 | 422.2 | +4.4 | 0.25 |

TABLE I

PERFORMANCE COMPARISON OF THE FOUR AES CONFIGURATION (I.E. DATAPATH). EACH CONFIGURATION CORRESPONDS TO A SPECIFIC TRADEOFF BETWEEN THE SECURITY LEVEL AND THE PERFORMANCE

Standards and Technology to replace the DES one. Furthermore AES is expected to be one of the major cryptography algorithm within IPSec which is a framework of different standards for ensuring secure private communications over the Internet. The major advantage of IPSec is its flexibility since it allows for negotiation of algorithm choices and configurations between the communicating parties. The algorithm parameters of AES are defined while the main mode and the quick mode security association steps of IPSec. The parameters negotiated in these previous phases and the current session keys are used to transmit data during the secure data transfer step.

All the experimentation has been conducted using a Xilinx Virtex-II Pro FPGA device [17]. Figure 1 presents the FPGA and the links between the processor and the memory that contains the different bitstreams (each bitstream corresponds to a configuration). The two registers within the SPC contain respectively the algorithm and architecture parameters. In our case the algorithm parameters are related to the type of algorithm (i.e. AES), to the execution mode of the primitive (i.e. feedback, non-feedback) and to the key and data sizes (i.e. 128 bits). The architecture parameters are focusing on the reliability (i.e. no, fault detection, fault tolerance), on the throughput, the area (use rate of the device) and the energy consumption.

In the following sections different points are analyzed. Section IV-A provides a comparison between several implementations of the AES datapath to define the performance and the costs of security. Then, section IV-B describes a bus monitor that tracks the accesses to the keys stored in the memory in order to detect hijacking. Finally, section IV-C discusses the efficiency of the whole AES security primitive.

### A. AES datapath implementations comparison

Three different datapaths have been implemented to show the flexibility provided within the primitive, feedback mode (FB), feedback mode with fault detection (FB_FD), and feedback mode with fault tolerance (FB_FT). An 128-bit key has been considered. Fault detection mechanisms enable the system to detect if a fault occurs during the computation of the AES algorithm but without correcting the result. A parity-based technique has been used to detect the fault [18]. Fault tolerance mechanisms provide a tamper resistant architecture. We have considered a TMR technique as it corresponds to a common solution [19]. Figure 4 illustrates the architectures of these primitives.

The implementations have been performed using the Xilinx ISE Foundation 6.3i tool and the power estimations have been done using the Xilinx XPower 6.3i tool. As shown in table I, each solution corresponds to different levels of performance in terms of area, throughput and power. Fault tolerance solution is the most secure one but the area and energy overheads are very high (respectively 6302 slices and 1673 mW). Fault detection using parity code does not lead to a significant difference in area and power consumption, respectively +2.1% of slices and -2.7% of power consumption compared to a non secured implementation in feedback mode. For these implementations the throughput is almost equivalent and around 400Mbits/s.

Another metric is interesting to compare these implementations, the energy efficiency which represents the throughput per energy (Gbits/J). Feedback with and without fault detection provide the same efficiency. Fault tolerance guarantees the security of the primitive but has a high overhead in energy efficiency. Thus, fault detection is a good compromise to guarantee the performance and to increase the security of the primitive and could be considered as an implementation by default.

### B. Bus monitoring

Tracking the activity on the bus corresponds to an interesting way to analyze the operation of the system. In our case we have defined a monitor that spies the address bus. As we can see on Figure 5 once the AES primitive starts the encryption, the accesses to the keys memory addresses are very regular. The first sequence corresponds to the generation of the sub keys from the cipher key. Then, each sequence represents the encryption of one block of data. As we can see 10 keys memory addresses accesses are performed for each block which correspond to the 10 rounds of the AES algorithm.

Two complementary scenarios have been considered to detect abnormal activity. the first one is based on a counter which compares the off-line profile with the run time memory accesses. For that purpose we have stored the off-line profile in a table (using a Huffmann-based coding to code the data) and we have implemented a counter that counts sequences of non-keys memory accesses and keys memory accesses. When

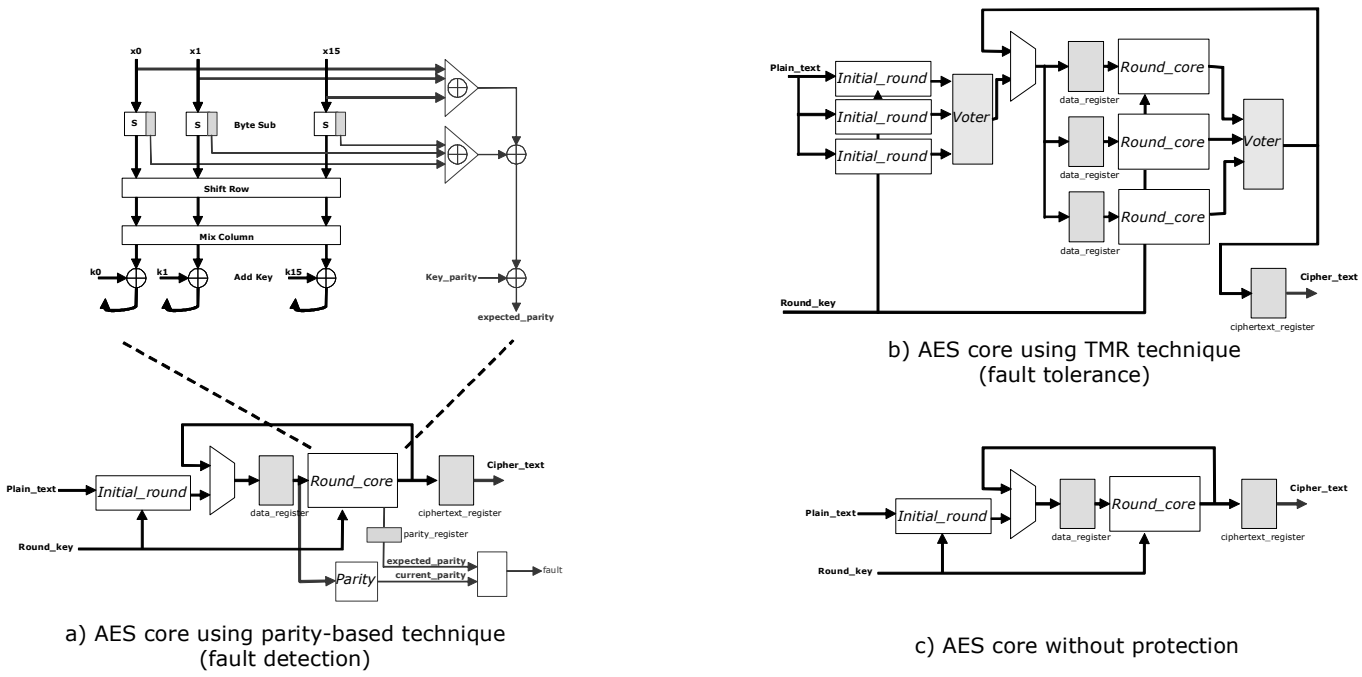


Fig. 4. AES core architecture for the three primitives: a)AES core using parity-based technique (fault detection), b) AES core using TMR technique (fault tolerance), and c) AES core without protection

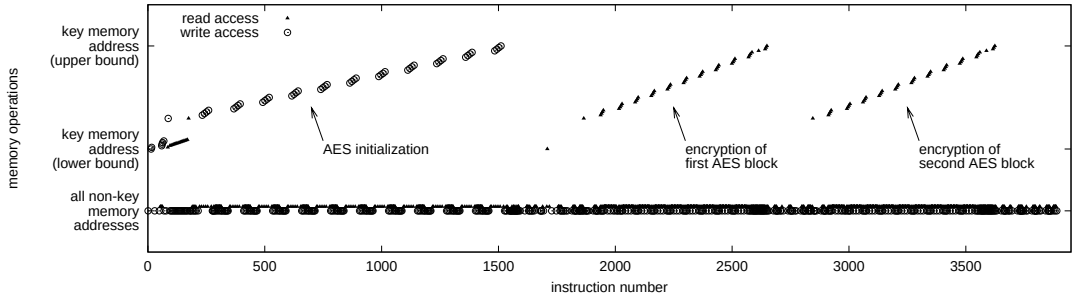


Fig. 5. Monitoring of the bus: accesses to the keys are highlighted which enables the monitor to detect some abnormal activities

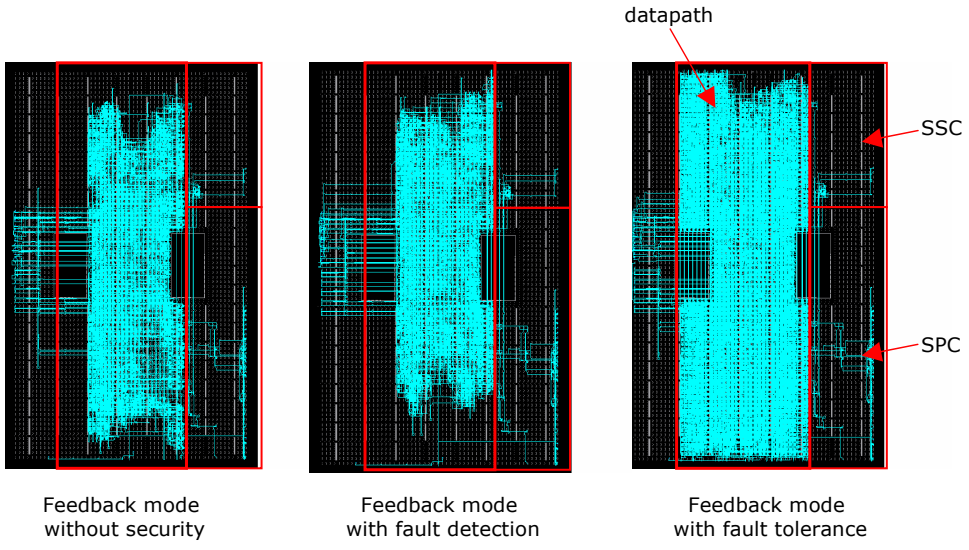


Fig. 6. Layout of the three configurations of the AES reconfigurable security primitive. Three modules are defined which are the datapath, the SPC and the SSC

the sequence matches what it is stored in the table, no alarm is raised. When there is a mismatch then there is a problem and the monitor indicates that there is potentially an attack. The monitor is flexible in the sense that the number of blocks to be encrypted is not known statically, so we have stored only one sequence that we compare as long as some blocks need to be encrypted. The second scenario is based on the combination of different data. Indeed, if the keys memory addresses are found on the bus and no encryption is running then it corresponds to an hijacking of the secret keys.

The complexity of the bus monitor depends on the monitoring technique as all source and destination addresses of reads and writes to/from keys memory can be analyzed. In our case we have considered a simpler solution as we only count the number of accesses and we do not consider the exact keys memory addresses. This solution leads to a small area overhead for the monitor but provides a less accurate approach. Dynamic reconfiguration of the monitor could be considered to adapt the accuracy of the monitor depending on the state of the system.

### C. AES reconfigurable security primitive efficiency

The three previous feedback implementations (FB, FB_FD, FB_FT) have been considered for the definition of the whole AES security primitive. We have defined three reconfigurable modules which are the datapath, the SPC and the SSC. An area constraint has been associated to each module as shown in figure 6. In this experiment we have considered a single primitive but there is no limitation concerning that point. The execution schedule between the processor, the SPC and the SSC is described in figure 7. It highlights when the reconfigurations occur. When the processor needs a security primitive, it configures the FPGA with the SPC and the SSC. The SPC indicates to the SSC that it has been configured and what function it has to realize. The SSC provides the SPC with data related to the battery state and the quality of transmission from the sensors. At the same time the SSC indicates to the SEP what type of primitive it is going to monitor, so that the SEP specialized it. Then, the SPC sends to the processor what type of configuration it can performs (mode, key size) based on the sensors information. Once the configuration is complete, the SPC is no longer involved in the datapath of the security primitive. However the SPC continues to poll via the SSC the state of the system to check if the mode of the security primitive needs to be changed (the aim is to change the mode if for example the battery is running low and the functionality is still to be available). At the same time, the SSCs are monitoring the system and if something abnormal occurs, then some modifications can be done (for example, to provide fault detection within the security primitive or fault tolerance)

The communications between the modules have been performed through 3 bus macro which are pre-defined Xilinx hard IPs [20]. One bus macro is used to provide the *fault* signal between the datapath and the SSC (figure 2). The two others are used between the datapath and the SPC and correspond to control signals (e.g. *start*, *reset*, *done*). The reconfiguration is performed by the SPC through the ICAP interface which allows the dynamic and partial self-reconfiguration of the FPGA [21]. Figure 6 shows the three possible configurations. As we can see, the area overhead for the fault tolerant implementation is high compared to the two other solutions. The SPC and SSC modules are very small and remain constant for the three configurations. Their complexity is small compared to the datapath so that they represent a negligible area overhead. For this study we have considered very simple performance and security policies which are basically based on a threshold crossing or on an attack or a fault detection. For real embedded systems, these policies might use more advanced techniques. However, the overhead costs should remain small compared to the datapath.

Concerning the performance of such a solution, the reconfiguration time is directly related to the size of the bitstream. The full bitstream which is used at power up represents 1415 kB and the three partial bitstreams for the FB, FB_FD, FB_FT configurations are respectively equal to 356 kB, 356 kB and 463 kB. In our case the clock of the ICAP interface is 50 MHz which leads to an average reconfiguration time around 8 ms. Each time a reconfiguration is performed there is also an overhead cost in term of power. However, this overhead is negligible for the FPGA power core and represents an increase of around 6% for the FPGA power supply [22].

## V. Conclusion and future work

We have presented the SANES architecture to improve security within embedded systems. The main concepts that drive the definition of this architecture is continuously monitoring the operation of the system to detect abnormal behavior and use of reconfigurable hardware to provide various levels of protection and performances. The combination of both approaches is to our knowledge an original work that enables the system to target both security standards and attacks. Results on the AES algorithm within IPSec show that the flexibility of our solution enables the definition of an energy-efficient solution while guaranteeing the security. Future work includes the definition of other monitors to detect attacks. This point is important as low complexity solutions have to be defined not to increase prohibitively the global cost of the system. Solutions based on signatures (using off-line profiling techniques) seem promising and will be further investigated.

## References


[1] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, *"Security in Embedded Systems: Design Challenges"*, ACM Transactions on Embedded Computing Systems, Vol. 3, No. 3, August 2004, Pages 461-491

[2] Gogniat G., Burleson W., and Bossuet L., "Configurable computing for high-security/high-performance ambient systems" accepted for the Embedded Computer Systems: Architectures, MOdeling, and Simulation Conference, Samos, Greece, July 18-20, 2005

[3] D. Nash, T. Martin, D. Ha, and M. Hsiao, *"Towards an Intrusion Detection System for Battery Exhaustion Attacks on Mobile Computing Devices"*, Proceedings of the 2nd International Workshop on Pervasive Computing and Communications Security, March 2005

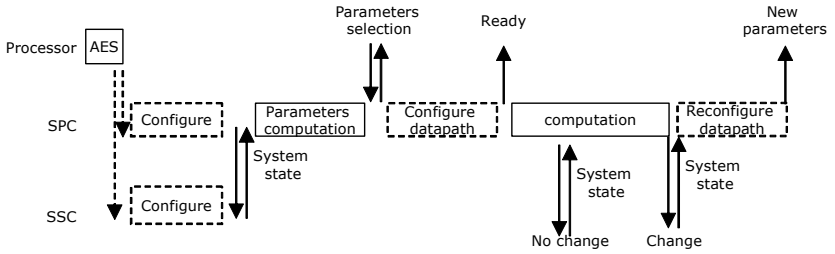


Fig. 7. Processor/security primitive schedule


[4] D. Lie, C. A. Thekkath, and M. Horowitz, *"Implementing an Untrusted Operating System on Trusted Hardware"*, 19th ACM Symposium on Operating Systems Principles, October 19-22, 2003, The Sagamore, New York, USA

[5] E. Suh, J. Lee, S. Devadas, and D. Zhang, *"Secure Program Execution Via Dynamic Information Flow Tracking"*, MIT, Memo-467, November 2003

[6] X. Zhuang, T. Zhang, and S. Pande *"HIDE: An Infrastructure for Efficiently Protecting Information Leakage on the Address Bus"*, in Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI) Boston, MA, USA, October 2004

[7] A. Hodjat and I. Verbauwhede, *"High-Throughput Programmable Cryptocoprocessor"*, IEEE Micro, May-june 2004, pp. 34-45

[8] D. Oliva, R. Buchty, and N. Heintze, *"AES and the Cryptonite Crypt Processor"*, In proceedings of CASES 2003, Oct-Nov 2003, San Jos, California USA

[9] P. Schaumont and I. Verbauwhede, *"Domain-Specific Codesign for Embedded Security"*, IEEE Computer, April 2003

[10] T. Wollinger and C. Paar, *"Security aspects of FPGAs in cryptographic applications"*, Chapter in "New Algorithms, Architectures, and Applications for Reconfigurable Computing", editors Wolfgang Rosenstiel and Patrick Lysaght, Kluwer, 2004

[11] A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, *"An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists"* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, volume 9, issue 4 (August 2001), pp. 545-557

[12] A. Dandalis and V.K. Prasanna, "An Adaptive Cryptography Engine for Internet Protocol Security Architectures" ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 9, N 3, July 2004, Pages 333-353

[13] E. Chi, A. M. Salem, R. I. Bahar, and R. Weiss, *"Combining Software and Hardware Monitoring for improved Power and Performance Tuning"*, The 7th Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT-7), Anaheim, California February 8, 2003

[14] J.S Seng, E.S. Tune, and D.M. Tullsen, *"Reducing Power with Dynamic Critical Path Information"* In proceedings of the 34th International Symposium on Microarchitecture, December 2001, Austin, Texas, USA

[15] T. Martin, M. Hsiao, D. Ha, and J. Krishnaswami, *"Denial-of-Service Attacks on Battery-powered Mobile Computers"*, Proceedings of the 2nd IEEE Pervasive Computing Conference, Orlando, Florida, March 2004, pp. 309-318.

[16] J. Daemen and V. Rijmen, *"The Design of Rijndael AES-The Advanced Encryption Standard"* Springer-Verlag 2002

[17] www.xilinx.com

[18] K. Wu, R. Karri, G. Kuznetsov, and M. Goessel, *"Parity Based Concurrent Error Detection for the Advanced Encryption Standard"*, International Test Conference 2004 (ITC), 2004, Charlotte

[19] C. Carmichael, *"Triple Module Redundancy Design Techniques for Virtex FPGAs"* Xilinx Application Note 197 (XAPP197) November 1, 2001

[20] Two Flows for Partial Reconfiguration: Module Based or Difference Based, Xilinx Application Note XAPP290, Xilinx, September 2004

[21] M. Ullmann, B. Grimm, M. Huebner, and Juergen Becker, *"An FPGA Run-Time System for Dynamical On-Demand Reconfiguration"*, The 11th Reconfigurable Architectures Workshop (RAW 2004), Santa F, New Mexico, USA, April 26 and 27, 2004

[22] J. Becker, M. Huebner, and M. Ullmann, *"Power Estimation and Power Mesurement of Xilinx Virtex FPGAs: Trade-offs and Limitations*, IEEE Symposium on Intregated Circuits and System Design, September 2003