Secure architecture in embedded systems: an overview

Romain Vaslin, Guy Gogniat, Jean-Philippe Diguet
LESTER UBS/CNRS FRE 2734
Rue de Saint Maud
BP 92116 - 56321 Lorient
firstname.lastname@univ.ubs.fr

***Abstract*— Security issues become more and more important during the development of mobile devices. In this paper we propose first a brief overview of hardware and software attacks related to embedded systems and second a comprehensive study of existing solutions to protect programs and data exchanges within these systems. Security primitives dedicated to the implementation of a secure architecture are also presented. Based on this analysis of existing solutions and requirements an original approach is proposed in order to mitigate the cost of security. Constraints related to embedded systems are strong it is thus mandatory to define new solutions, our proposition is outlined through various security primitives (ciphering and hashing) with features adapted to embedded systems.**

I. Introduction

With the development of new wireless communication standards like WIFI and Bluetooth, the communications between entities (cell phone, PDA) is becoming unavoidable. Sometimes sensible data is exchanged (e.g. credit card number); so it is necessary to protect these transfers. Security is turning into the main bottleneck for communicating entities especially in embedded systems where performances are limited. More and more systems are facing hardware and software attacks [8]. Several solutions are proposed to protect the architecture (secure architecture) and the data which is transferred (cryptography). Architecture protection mainly corresponds to the protection of data and program stored in the system memory. Communication protection is related to the protection of data exchanged over an insecure communication channel (e.g. wire).

When a system is under attack, different goals are targeted, the first kind of attack is the extraction of secret information, the second one is trying to put the system out of order. Security is based on five essential principles which are supposed to guarantee the correct execution of both the program and the communication:

- Confidentiality: only the entities involved in the execution or the communication can have access to the data,
- Integrity: the message must not be damaged during the transfer or the program must not be altered for the execution,
- Availability: the message or the program must be available,
- Authenticity: the entity must be sure that the message comes from the right entity or the system must trust the program source code,
- Non-repudiation: the entities implied in the exchange must not have the possibility to deny the exchange.

Cryptography corresponds to a partial solution to these issues. The encryption of information is used for confidentiality. For example, only the users with the encryption or the decryption key are able to communicate together. The most popular cipher algorithms are: RSA [1], ECC [2], AES [3], 3DES [4]. RSA and ECC are asymmetric cipher algorithms. In this case the key used to encrypt (public key) the message is different from the key used to decrypt the message (private key). As the key used to encrypt the message is public, everyone can send a ciphered message to the entities which own the private key to decrypt the message. AES and 3DES are symmetric cipher algorithms. The key used to cipher must be secret because it is the same key for encryption and decryption.

The hash of information is used to check the integrity of a message by providing a signature which is unique for each message. The most known algorithms are MD5 [5] and SHA [6]. The robustness of the SHA family varies according to the number of bits used for the coding of the signature. In addition, non-repudiation, availability and authenticity are guaranteed by communication protocols like IPSec for example [7].

More and more security tasks are assigned to embedded systems. Thus, it becomes interesting to add dedicated primitives to these systems to allow an efficient implementation of the requested algorithms for program and data protection. As a consequence, various solutions are emerging to increase the system protection. It is essential that these solutions provide hardware architectures adapted to embedded systems. Classical solutions from computer science do not answer the problem. Many constraints are due to the application and environment requirements (memory size, performance, power consumption).

In the following sections we propose first a brief overview of existing attacks towards embedded systems (hardware and software attacks). Second, a state of the art of current solutions to protect a system and to speed up cipher algorithms is provided. Then, the outline of an original approach based

on configurable hardware to accelerate cipher algorithms is presented.

II. Hardware attacks

The main goal of hardware attacks depends on the wish of the attacker. Two main opportunities can be targeted. The first one is trying to get secret information like cipher keys. The second one is to attack the system to turn it out of order (i.e. denial of service attack). Below attacks which aim to catch secrets are presented, then denial of service attacks are detailed. Some attacks are difficult to classify, hardware modification of the main memory is one of them. The goal of this attack is to insert a malicious program. A similar attack targets FPGAs through bitstream alteration.

When the attacker wants to decrypt information, he needs to have the cipher key. A solution to get cipher keys is to listen to side channels. This kind of attack is called *side channel attack* and is declined in several forms [9]. The most known relies on the power signature of the algorithm [11]. By analyzing the algorithm signature it is possible to infer the round of the algorithm. Moreover, a differential analysis combined with a statistic study of the power signature can lead to an extraction of the cipher key [11]. However it is necessary to make assumptions on the value of the key to obtain a correct result. These two methods are called SAP: *Simple Power Analysis* and DPA: *Differential Power Analysis* [11]. Similar solutions also work with electromagnetic emissions [12] (*Differential Electromagnetic Analysis*). Instead of analyzing the power signature, the electromagnetic signature of the chip is analyzed. A significant remark concerns the cost of such attacks. It is especially cheaper than reverse engineering attack which needs an electronic microscope to study the structure.

Temporal analysis or timing attack [13] is another way to catch cipher keys. Temporal reaction of the system leaks information which enables the extraction of cipher key or password. Like with the DPA, it is necessary to make assumptions concerning the information to be extracted. The knowledge of the algorithm, so the branch instructions in the program can also help to find a secret since a timing model of the algorithm can be established [13]. Indeed, timing hypotheses can be done as the program running on the target is often known. Thus, thanks to statistic studies, information can be extracted.

Fault injection [14] is the last way to obtain secrets through side channel. However, like reverse engineering, the need of material is more important than previous attacks. The injection of a fault into a system through a memory corresponds to a modification of a bit (laser or electromagnetic waves). The knowledge of the implementation of the algorithm is an important point to determine a secret. In most cases the injection of a fault is done in the last round of an algorithm [14]. The reason is that the mark of the fault is more visible in the ciphered result.

The goal of the hardware attacks presented above, is to get secret information from the chip. *Denial of service* attacks are different and aim to put the system out of order. In autonomous embedded systems, power is an essential concern. It is one of the most important constraints on the system. As an example with a cell phone or a PDA, the attacker can perform a large number of requests which aim to activate the battery and to reduce the system lifetime [15] [16]. In wireless communication systems, another attack leads to solicit the transmitter antenna in order to have the same result as previously (lifetime reduction). Increasing the workload of a processor is also an issue to consume more battery. Indeed the workload is related to power consumption, so an assailant may try to force the processor to work harder [16] [15]. As a consequence the lifetime will be affected. Other ways can be used to put a system out of order. Taking the control of the temperature regulation system is a solution. Through the control of the regulation it is possible to increase the temperature and then to activate the overheat security mechanisms [17].

The panel of attacks against a system is important and depends according to several parameters: goal, budget and nature of the system. Hardware attacks represent an important threat against embedded systems but software attacks are also becoming critical.

III. Software attacks

Like computer science (server and workstation), embedded systems are more and more affected by virus and worms [8]. There is a difference between a virus and a worm. We can consider that a virus needs the human help to infect a system and to spread contrary to a worm which does not need any human help. A worm is considered to be autonomous. All the computer science concepts can be transposed to the embedded system domain. The substitution of a program by a malicious one is a threat for the security of the system. The malicious program may try to get access to sensitive data or to shut down the system. Concerning secret data, cipher keys are the most sensitive as once the attacker knows the cipher keys, he has access to all the information in plain. Encrypting memory and protecting cipher keys correspond to classical solutions against these attacks. However protections used in computer science are not suited for embedded systems (less computing power and memory). Thus various solutions dedicated to embedded systems are emerging (e.g. bus or program monitoring) [18].

The number of attacks targeting embedded systems increases rapidly. For example a virus or a worm can be sent several times on a same system to launch the antivirus. Scanning all the system increases processor workload and thus decreases the battery lifetime which can be critical for autonomous systems. The concept of embedded systems extends the scope of activity of the virus and the worms.

IV. Secure architectures: state of the art

In order to fend off previous hardware and software attacks specific mechanisms have to be defined. All security solutions are built around assumptions concerning their potential threats. For example in Figure 1, the secure zone is composed of the processor core and the ciphering and hashing dedicated blocks. Ciphering and hashing protection methods are similar to the

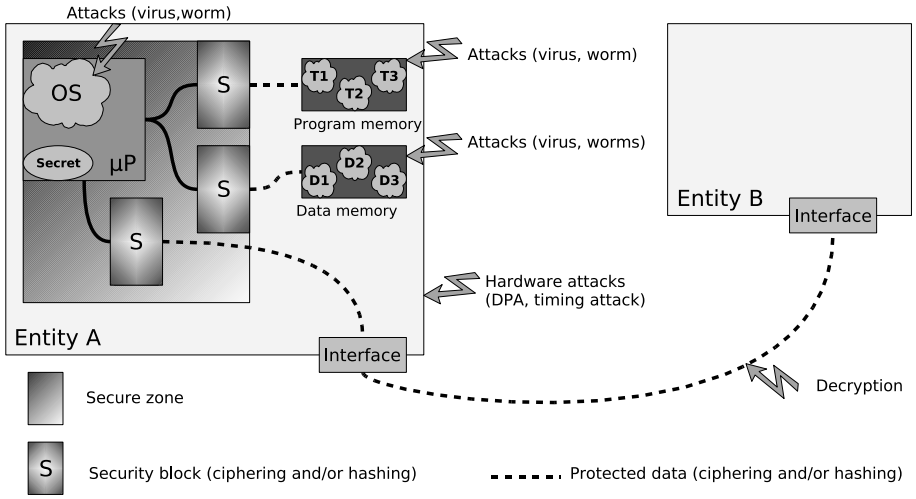

Fig. 1. Architecture with a secure core and secure communication

ones used for protection of a memory or a communication over a non-secure channel. In the following sections, the secure architectures consider that the core and the blocks are in a secure zone (i.e. cannot be attacked). Furthermore, these architectures do not consider side channel attacks.

In section IV-A the studies focus on the protection of program memory and data memory. Moreover, a monitor is used to protect the operating system (OS). Using an OS, there is a need for the system to track if a task does not reach any secure information not belonging to it. In a same way, before running a task, the OS must allow or not the task. In other words, is there an external entity which has altered the original task? In certain circumstances, the user may wish to cipher and/or hash the program in memory. Then if the program is read in the memory, the cipher key will be necessary to decrypt the data. As the cipher key is a secret and stored in the secure zone, only a trust task must be able to decrypt the program and to run it on the OS. As shown in Figure 1, the secrets stored on a chip are always in the secure zone. It is one of the most essential postulate when defining a secure architecture (the secret must not leave the secure zone in a clear form). The secrets may be the cipher keys but also the boot program of the application running on the chip. With an OS, the OS source code will be stored in the secure zone since it is essential that the OS kernel is not corrupted by a malicious entity. The solutions in section IV-A are built based on these mechanisms. OS solutions are generally associated with cryptography hardware engines and specific OS primitives to use these engines. In section IV-A we focus on the principles of the OS and not on the hardware engines to accelerate the computing of the cryptographic tasks. Section IV-B details the hardware engines to efficiently implement encryption, decryption and hashing functions for embedded systems. The (re)configurable architectures presented in section IV-B may be used by the secure architectures of section IV-A to speed up the work of cryptography primitives. As shown in Figure 1, the blocks used for the security of the memory are the same ones as the blocks used for the security of the external communications. Cryptographic hardware engines are preferred to software solutions since performance are better. The last point highlighted in section IV-B is related to the integration of the engines within the architecture (coprocessors or accelerators).

A. Program and data security software-based solutions

1) Trustzone [19]: Trustzone is a solution proposed by ARM. ARM considers that the complete secure solution is not feasible and targets to secure only some parts of the architecture and some data. Like other solutions, Trustzone postulate is an architecture with a secure core and a secure part within the memory (secure zones in Figure 2). An important point is that Trustzone does not provide any mechanisms for cryptographic issues. If a user wishes to cipher and/or hash some data, he has to develop the corresponding software or hardware security primitives.

The guiding principle of Trustzone is to add an extra mode (secure mode) to those already known (user, superuser). A monitor supervises all the operations of the OS and especially when an application is switching from/to the secure mode. The monitor allows or not the switching from one mode to another. Once the application is running in the secure mode, the user can have access to all the protected data and programs stored in the memory. As an example the cipher keys and the boot program are considered as sensible data. When the secure mode is active, the monitor supervises all operations to be sure that a task which is not allowed is not trying to catch illegal information.

The most significant part of the work for the monitor is to protect the accesses to data. Several hardware mechanisms have been added to the architecture to support this feature. The Trustzone architecture proposes cache memories and uses a memory management unit to provide a more efficient solution. Thus, some modifications have been performed to support the new possibilities of the architecture. They enable the monitor to be informed if an access to a protected data is

done or not. Some peripherals can also be included in the trust zone, thus specific methods are required to protect the communications with them. In Figure 2, the secure and share zones correspond to an example. Concerning the external memory, ARM suggests to cipher and to hash it. The attacker will not be able to interpret the data and program because he does not have the cipher keys. In a same way, the hash of the memory helps the architecture to keep the integrity of the source program and data. Moreover, since some peripherals are included in the secure zone, the communications between the peripherals and the core require new signals to exchange data.

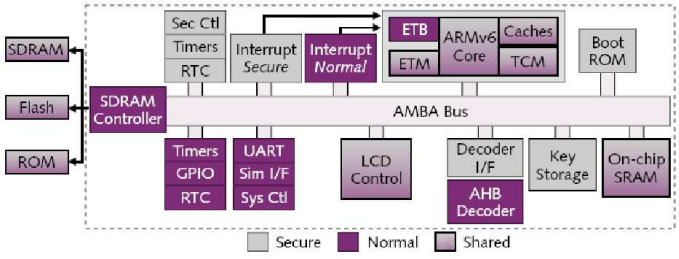

Fig. 2. Secure architecture of Trustzone [19]

2) XOM [20]: XOM is the acronym of eXecute Only Memory. XOM wishes to completely secure an architecture. Moreover XOM is supposed to be sure and claimed that hardware solutions are more efficient than the software ones. So XOM mostly relies on hardware mechanisms to ensure security. New primitives are provided within the OS in order to handle key and signature manipulation. The name of the OS extension is XOMOS. The main features of XOM are: memory ciphering and hashing, data and program partitioning, interruption and context switching protection.

Each partition of the memory is associated with a secret key to decrypt its content (the session key in Figure 3). The session key is obtained with the XOM key table which establishes the connection between the session key and the secret key of a specific partition of the memory. The secret key is also encrypted with an asymmetric encryption. The key (private key in Figure 3) required for the asymmetric decryption is stored in the secure zone of the architecture. The use of the hash solution is classic. The signature result of the hash algorithm is compared with the original one to validate the integrity of the hashed message. In addition the data stored in cache memory is associated with an identifier. When a task wants to use a data, the identifier of the task must be the same as the data one, in that case it means the task is allowed to read and modify the data. This feature protects the system from malicious program which tries to get illegal information. XOM proposes hardware security primitives to protect cipher keys and hash signatures which are essential to guarantee the architecture durability.

The last point of the XOM solution concerns the preemption within the OS which has similarities with the management of the interruptions. The context must be saved. It is essential to store and to protect the context in order to fend off an attack who aims to change some register values. XOM ciphers and hashes the switching context which is interesting for a solution with an OS. XOMOS can be seen as an extension of a non-secure OS which brings new security primitives (ciphering and hashing).

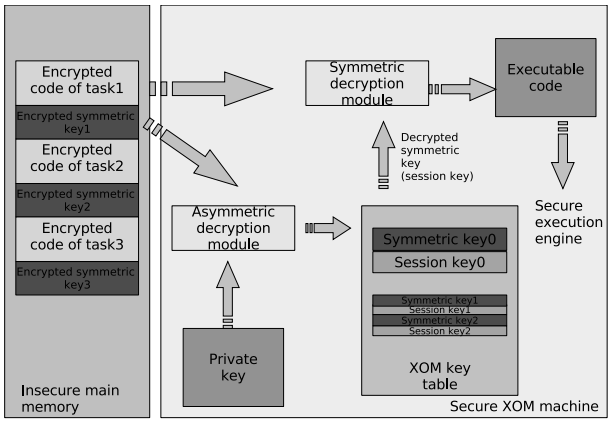

Fig. 3. XOM architecture [20]

All the protections added by the solution have a cost. The first one concerns the implementation of XOM in an existing OS. A work is necessary on the kernel to add the instructions which help for the use of the security primitives. All this work is invisible for the user of the kernel. A real overhead appears in the cache management. The number of cache miss raises from 10 to 40%. It depends on the kernel operation. This raise is due to the information added in the cache to secure the data. Data are associated with the identifier of the task. It means some parts of the cache are used to store the identifier. The protection of the context switching also brings an increase of the number of cycles to store the context and to protect it.

3) AEGIS [21]: AEGIS is an OS solution like XOM. Figure 4 shows the secure computing model of AEGIS. AEGIS is based on a postulate concerning the threats. The grey parts in Figure 4 corresponds to secure zones that are supposed to be protected by default. As very often the memory and the cache memory are not included in the trust zone. The components required to build the security primitives, are considered to be secure. The main features of AEGIS are: generation of secret with PUF (*Physical Random Function*), memory protection by ciphering and/or hashing, variation of the level of kernel security.

The PUF is an hardware mechanism which provides an unique secret associated to a chip. The propagation time within the chip corresponds to the base of the PUF. PUF is a random source used to create the secret which is based on a sequence of multiplexer giving a bit as a result. The fabrication process of integrated circuit (IC) is the source of the uniqueness of the propagation delay (each IC has its own delay). The sequence of multiplexer makes the chip unique and the result of the sequence is very difficult to predict. A regulation system is required to limit the variation in the

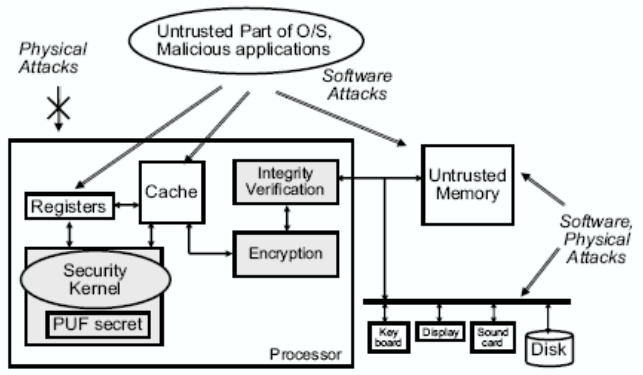

Fig. 4. Security model of AEGIS [21]

result of the sequence since the result sent by the PUF must always be the same (required for cryptography). Moreover, PUF is associated with a hash algorithm to increase the complexity of the secret generation.

Memory protection is an important point as the memory corresponds to a non-secure zone of the architecture. Thanks to the secret obtained with the PUF, the data and memory are ciphered and/or hashed. Furthermore, the memory security is also obtained through the MMU (*Memory Management Unit*) which manages the security levels of the workspaces (user and superuser, secure or not). Each user can choose to cipher (or not) and to hash (or not) the data. Thus AEGIS provides the mechanisms to choose the level of security of a piece of program. For example the boot program can be ciphered and hashed for more security.

AEGIS seems to be a very complete solution to protect memory and program. The overhead is important in some domains. The silicon area is one of them. It is increased by 1.9 [37]. The CPU core is the part which is the most affecteded by this overhead. Moreover, all the logic needed to control the specific mechanisms contributes to raise the area. The global performances of the architecture depend on certain parameters like the sizes of the protected memory and the cache memory. The workload varies according to the chosen security primitives which means the processor workload is directly linked with the security policy.

B. (Re)configurable hardware architectures

This section details main trends concerning hardware approaches to implement encryption, decryption and hashing functions in an efficient way for processor-based embedded systems. Hardware security engines can be subdivided in three categories: coprocessors, accelerators and dedicated processors. A coprocessor is implemented in the datapath of a processor contrary to an accelerator which is connected as a peripheral through a bus. A coprocessor is accessible through registers like an ALU. A dedicated processor is a processor with specific security features (e.g. hardware hash engine). Coprocessors and accelerators can be divided in two classes depending on their execution model since the (re)configuration can be performed at design time or at runtime.

1) Dedicated processors: A dedicated processor implements specific instructions dedicated to security primitives. An analogy can be done with DSP through its multiplication-accumulation instruction for digital signal processing. In most cases, security processors are dedicated to one class of cipher algorithm (symmetric or asymmetric). Specific execution units are included in the datapath. [22] and [23] propose processors with instructions for symmetric cipher algorithms. Specific instructions have been defined like logical operation (xor-add) or data permutation. For processors dedicated to asymmetric cipher algorithms [24], specific instructions are defined to efficiently compute the modular exponentiation which is an essential operation (ECC and RSA).

2) (Re)configurable architectures at design time: Architectures (re)configurable at design time offer an higher level of flexibility compared to dedicated processors since they provide several modes of execution. [26] and [25] propose two hardware accelerators in order to speed up ciphering operations. Their architecture is fixed and controlled through configuration registers. The main feature of [26] is its ability to run several algorithms in parallel and to select the execution parameters associated to each security primitives. [25] is a configurable solution which allows the user to switch in different modes of the AES algorithm [3]. In both cases the architecture is dedicated and optimized for an algorithm.

Another approach consists is specializing the architecture during the compilation step to produce an efficient secure architecture dedicated to the application. First solutions using such a technology were not dedicated to security [27]. In [27] the authors propose an architecture with the possibility to choose the execution unit within the core of the processor. The drawback with this approach is that the user is strongly involved in the development process to identify the right functionalities. An evolution of this solution in the domain of digital signal processing is XiRisc [28]. The processor core is fixed and connected to a reconfigurable coprocessor. After analyzing the program, main characteristics are extracted to implement some specific functionalities in the coprocessor. The result for the architecture is some new instructions specific for the application. With XiRisc the reconfiguration is done when powering up the architecture. Such features are very interesting for embedded systems and have been extended to the security domain. Furthermore, the results obtained with this solution are really interesting as for an implementation of DES algorithm, the speedup is about 13 times compared to a non reconfigurable solution.

In [29] the authors have considered a similar approach for security applications. By exploiting the Xtensa architecture of Tensilica [30], the authors show that the performances of security primitives (ciphering, protocol) are strongly improved (65% for MD5 and 75% for AES). The improvement is due to the coprocessor connected to the Xtensa architecture. Like XiRisc, the largest part of the design is done at compilation

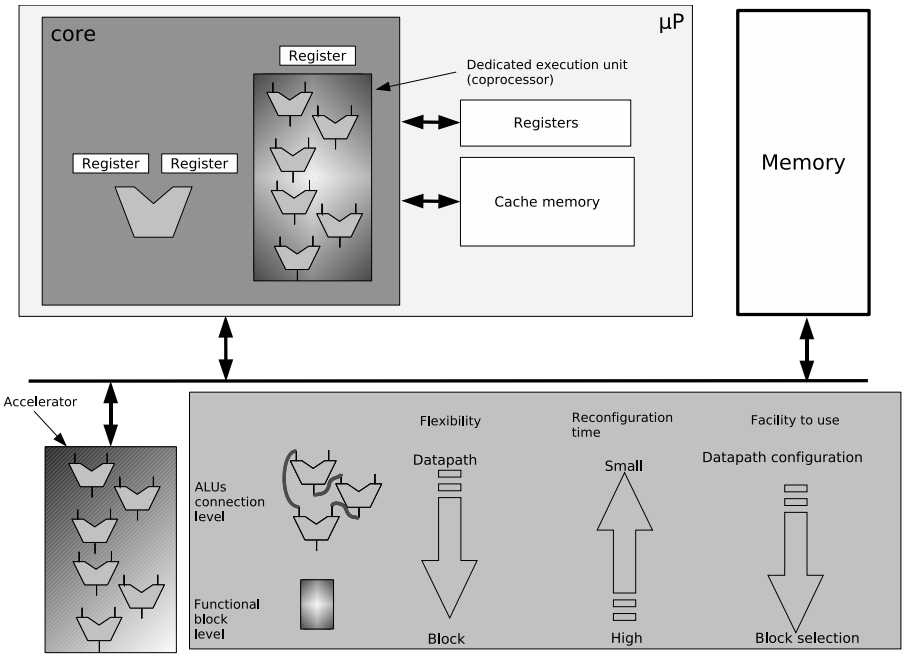

Fig. 5. Architecture with a coprocessor and an accelerator

time. The analysis is performed during compilation and the reconfiguration is done at power-up of the architecture. Specific tools for the architecture are required to build an efficient solution (compiler, linker or simulator).

3) (Re)configurable architecture at runtime: (Re)configurable architecture at runtime is an interesting alternative since the datapath can be adapted dynamically in order to provide the right security primitives depending on the requirements (e.g. hashing, ciphering). Compared to previous solutions this approach offers the highest level of flexibility and provides very efficient solutions. As detailed hereafter this solution is very interesting for embedded systems, unfortunately no work has been reported in the security domain. However in this section a description of this technology is still provided as we believe similar secure architectures should appear in a near future. The base of this approach is to reconfigure a coprocessor during the program execution when the logic is unused. In [31] the architecture core is fixed and the coprocessor can be dynamically reconfigured. As the previous solutions, a work is necessary to adapt the program of the application in order to take benefit of the coprocessor. The main difference comes from the reconfiguration model. If the logic associated with a specific instruction is not loaded into the coprocessor when required then the reconfiguration is performed dynamically. The reconfiguration only affects the datapath and not the ALUs within the coprocessor (coarse grain reconfiguration). The reconfiguration helps minimizing the silicon area of the chip to improve the power consumption and to provide efficient execution patterns to speed up the execution. At design time dedicated tools are required to define specific instructions and the logic of the architecture. In [38], it is shown that the reconfigurable coprocessor speeds up the architecture by 190 for a specific application (EEMBC). If the application is implemented with instruction reliing on the coprocessor, interesting results can be obtained.

A similar approach is proposed in [32] where the authors define a complete reconfigurable core for the processor. The instruction set of the architecture is fixed but the core of the processor has different configurations for the ALU. The reconfiguration of the block is done at runtime depending on the instructions to be executed. The decision to reconfigure (or not) comes from the pipeline stages: fetch, the cache trace and eventually the prefetch. An interesting point concerns the compilation. For this architecture there is no need for a special compiler as the instruction set of the architecture is not modified for each application. The processor dynamically configures its datapath to increase its performances. Similar concepts can be considered for the security domain in order to build a processor-based solution relying on a dynamically reconfigurable datapath (coarse or fine grain).

4) Limitations of existing solutions: Hardware solutions presented above are not always targeting embedded systems which involve very tight power consumption and small silicon area. Using an hardware accelerator [26] [25], leads to high performances but at the cost of power consumption which can be prohibitive in some cases.

In the case of configurable architecture [29] several remarks can be done. This approach is strongly adapted to embedded systems as it minimizes the power thanks to configurable features and improves the performance due to specific instructions. The most important concern is related to the development process which can be tedious in order to define the right instructions. It is essential that the architecture

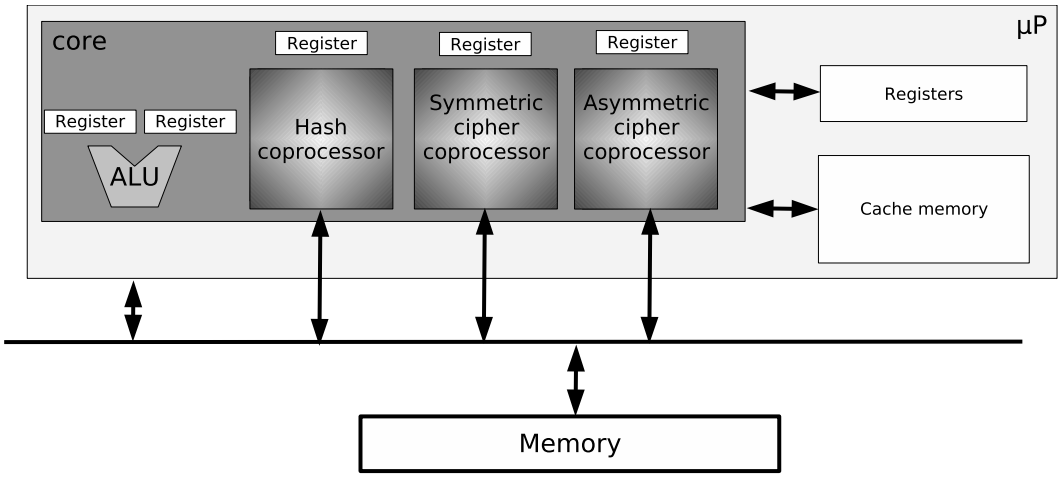

Fig. 6. Architecture proposed

supplier provides an efficient compiler which can identify and exploit specific instructions. For architectures like [31], when extended to the security domain, the difficulty will rely mainly on the definition of the reconfigurable datapath (granularity, flexibility). The users must have a deep understanding of the architecture and its basic datapath in order to extend and optimize the execution units.

As shown in Figure 5, reconfiguration of the ALUs interconnections leads to very flexible architecture. The user has the ability to build efficient ALUs by configuring the datapath. However, if no tools are provided with the architecture, this task may be tedious since the user has to know the ALUs implemented in the logic to develop his own security functions. Datapath reconfiguration is interesting since it corresponds to an efficient tradeoff between flexibility, reconfiguration time and performance. Block reconfiguration provides an higher flexibility but at the cost of reconfiguration time (issue of granularity vs. efficiency). This disadvantage is mitigated by the fact that the system becomes simpler to develop since it is mainly based on security IPs, thus the designer does not need to have a deep knowledge of the security cores. Coarse grain coprocessors based on datapath reconfiguration are more complex to develop as the designer needs to defined all the execution patterns that will be implemented in the datapath.

Both solutions provide interesting features, thus defining an architecture corresponding to a compromise between these two approaches needs to be evaluated. Moreover, it is essential to keep in mind that tools allowing the efficient use of the architecture are mandatory (compiler, simulator) to provide a comprehensive solution. Next section addresses this issue and proposes the outline of a configurable processor-based solution dedicated to security primitives.

V. Configurable coprocessor-based architecture

A. Toward a compromise between flexibility and programmability

The main objective of our approach is to combine both 1) the implementation simplicity through the use of an architecture with (re)configurable functional blocks and 2) the flexibility where interconnections between ALUs can be (re)configured within a functional block. Obviously compromises are necessary since both points are not entirely compatible. To propose a relevant solution the number of configurations needs to be limited. In practice some cryptographic algorithms are mainly used: MD5 and SHA for hashing, 3DES and AES for symmetric algorithms, RSA and ECC for asymmetric algorithms. The goal is to define a (re)configurable architecture dealing with these algorithms. Each algorithm can be associated to a dedicated coprocessor (as shown in Figure 6) with specific instructions.

The proposed coprocessors could be used within systems like in section IV-A to speed up the cryptographic primitives. Thus, blocks ciphering and hashing of the entire memory could be achieved more efficiently. Three coprocessors can be run in parallel to take benefit of the potential parallelism between cryptographic operations (memory hashed and ciphered). The cost of one coprocessor will be low due to common ALUs sizes between algorithms. For each cryptographic family the flexibility will be limited through the use of a coarse grain configurable ALUs. Various examples will be presented in section V-B with the hash coprocessor.

It is important to define an architecture which can be programmed efficiently which constrains the flexibility of the architecture. The use of the coware LISAtek tool suite [33] in interesting in order to build a processor-based system and the associated compiler. It will enable the designer to quickly develop an architecture but also to produce all the tools required for its use in order to exploit the possibilities of the architecture [34].

B. Coprocessor dedicated to hash case study

In order to demonstrate our ideas an hash coprocessor is considered. We aim to build a coprocessor for the MD5 algorithm and SHA family (SHA-1 and SHA-2). Within these algorithms, two specific stages are performed during the execution. The first one is the message preparation and the second

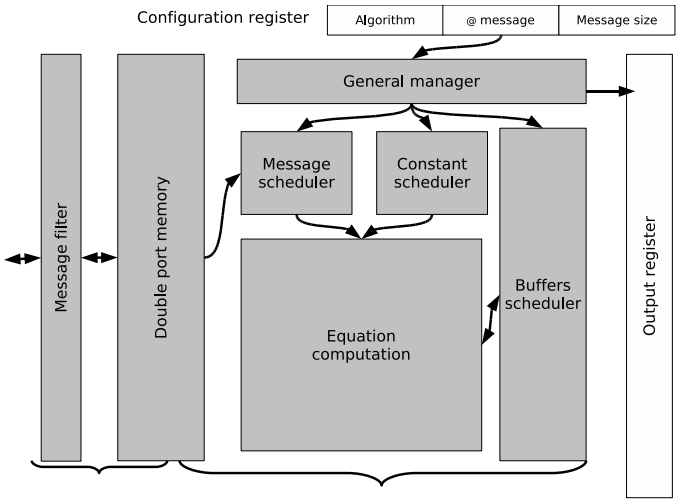

Fig. 7. Hash hardware dedicated coprocessor

equation	SHA-1	SHA-2	MD5
$(x \wedge y) \oplus (\bar{x} \wedge z)$	x	x	x
$x \oplus y \oplus z$	x		x
$(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$	x	x	
$rot^m x \oplus rot^n x \oplus rot^l x$		x	x
$(x \wedge y) \oplus (y \wedge \bar{z})$			x
$y \oplus (x \vee \bar{z})$			x

TABLE I

SUMMARY OF EQUATIONS FOR DIFFERENT HASH ALGORITHMS

one is the hashing of the prepared message as illustrated in Figure 7 (Part 1: message preparation, Part 2: message hashing). It appears natural to divide the architecture in two dedicated blocks. The message filter is configured depending on the selected hash algorithm (datapath size, message size).

Concerning the part 2, a deeper analysis of the hash algorithms is required to find similarities. Due to the complexity of the algorithms, the number of specific ALUs may increase since it becomes more difficult to find similarities. However, it is possible to extract some common functional blocks from algorithms as in Figure 7. A minimum memory (1024 bits) is required to store some data before the hashing step especially for SHA-512 and SHA-384 which need an important amount of memory (before starting to hash, a message of 1024 bits is required).

A paramount element of the architecture is the block for the equation computation (selection of the right equation among the available equations). A thorough analysis is required to find the similarities between all the equations of the whole hash algorithms. Work presented in [35] can be used since the authors show that similarities can be found for certain hash algorithms (MD5, SHA-1). This work leads to the definition of a configurable functional block for the computation of the equations. With [10], it is possible to extend the work of [35] to the whole SHA-2 family. In Table I, some similarities between the equations are presented. The minimization of the coprocessor can be performed based on these similarities in order to minimize the power consumption. Equations in Table I highlight that it is possible to find similarities at a coarse grain for the ALUs. In [35] the authors propose a customizable function depending on the hash algorithm. The proposed approach can be extended to all the algorithms within our hash coprocessor.

The concepts of functional block can be further extended since the selected hash algorithms have similarities which allows to create other blocks like buffers scheduler, constants scheduler and a message scheduler. The proposed architecture is presented in Figure 7. Each one of the scheduler provides inputs to the equations block. The supervisor controls all the schedulers of the architecture and is in charge of the general management thanks to the information given by the configuration register. The register contains data which are read by the general manager to configure the architecture to correctly execute the right hash algorithm. The result for the user is just a value to write into the configuration register.

C. Discussion

This section outlines a configurable coprocessor-based approach which must be pursued to obtain more results and to extend it to others coprocessors (symmetric and asymmetric ciphering). Certain similarities concerning asymmetric ciphering can already be defined as the modular exponentiation operation is used in both computation (RSA and ECC).

Two approaches also need to be further explored. The first one concerns the grain of the ALUs. Finer grain for the ALUs may be identified to better reduce the area required to implement the algorithms. APIs may be also developed to manage the configuration of the coprocessor. The user should be able to use these APIs to configure the coprocessor. The goal is to mitigate the work required to use the configurable coprocessor. The second point that needs to be explored, is the implementation of different modes for an algorithm. For example with an AES ciphering, a fault tolerance mode may be implemented [36]. It is also important to define the mechanisms in order to be able to dynamically adapt the configurable coprocessor in order to adapt its datapath depending on the requirements and the constraints on the system.

VI. CONCLUSION

Hardware approaches within secure embedded systems represent a very interesting solution to increase the protection of programs and communications while reducing the cost of security. Standard solutions from computer science are not directly suitable and must be adapted to embedded systems domain. Furthermore embedded systems are facing more and more attacks tacking benefit of the constraints related to their domain. It is thus necessary to define new techniques to protect these systems.

In this paper we have proposed a state of the art of emerging technologies used in order to increase the protection of these systems at the software and the hardware levels. We have also defined some rules in order to improve the performance of the

security primitives. It is thus essential to provide new hardware engines (ciphering/hashing hardware) adapted to embedded systems constraints before building a complete secure architecture (core, memory). (Re)configurable solutions provide some interesting features that should be better analyzed in order to promote the flexibility but also the programmability. It is also important to study fine grain techniques in order to fend off some specific hardware attacks.

Finally we have presented the outline of our approach to build hardware engines required for a secure architecture. We have focused on the ciphering and hashing architecture for embedded systems. Future work will target a more precise evaluation of our approach to evaluate the achievable performances and the efficiency of the programmability.

REFERENCES

[1] RFC 2313, ftp://ftp.rfc-editor.org/in-notes/rfc2313.txt
[2] RFC 3278, ftp://ftp.rfc-editor.org/in-notes/rfc3278.txt
[3] RFC 3565, ftp://ftp.rfc-editor.org/in-notes/rfc3565.txt
[4] RFC 1851, ftp://ftp.rfc-editor.org/in-notes/rfc1851.txt
[5] RFC 1321, ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt
[6] RFC 3174, ftp://ftp.rfc-editor.org/in-notes/rfc3174.txt
[7] RFC 2401, ftp://ftp.rfc-editor.org/in-notes/rfc2401.txt
[8] D. Dagon, T. Martin, and T. Staner, Mobile Phones as Computing Devices: The Viruses are Coming!, IEEE Pervasive Computing, 2004
[9] Guilley, S., Pacalet, R., Soc securiy: a war against side-channels. In of the Telecommunications, A., ed.: Systeme sur puce electronique pour les telecommunications. 2004
[10] FIPS 180-2, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[11] Paul C. Kocher, Joshua Jaffe and Benjamin Jun, Differential Power Analysis, Proceedings of the 19th Annual International Cryptology
[12] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, Pankaj Rohatgi, lecture Notes in Computer Science Multi-channel Attacks 2003
[13] Paul C. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology 1996
[14] P.-Y. Liardet and Y. Teglia, From Reliability to Safety, workshop on fault diagnosis ans tolerance in cryptography, 2004
[15] Daniel C. Nash et all, Towards an Intrusion Detection System for Battery Exhaustion Attacks on Mobile Computing Devices, PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops,2005
[16] Thomas Martin et all, Denial-of-Service Attacks on Battery-powered Mobile Computers, RCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications, 2004
[17] Puyan Dadvar and Kevin Skadron, Potential Thermal Security Risks, 21st IEEE SEMI-THERM, 2005
[18] Joel Coburn et al, SECA: security-enhanced communication architecture, international conference on Compilers, architectures and synthesis for embedded systems, 2005
[19] ARM trustzone http://www.arm.com
[20] XOM project: http://www-vlsi.stanford.edu/ lie/xom.htm,
[21] AEGIS project: http://publications.csail.mit.edu/abstracts/abstracts05/suh/suh.html,
[22] Rainer Buchty, Nevin Heintze, and Dino Oliva, Cryptonite A Programmable Crypto Processor Architecture for High-Bandwidth Applications, 2004
[23] Lisa Wu, Chris Weaver and Todd Austin, CryptoManiac: a fast flexible architecture for secure communication, ISCA '01: Proceedings of the 28th annual international symposium on Computer architecture, 2001
[24] Hans Eberle et all, A Public-Key Cryptographic Processor for RSA and ECC, ASAP '04: Proceedings of the Application-Specific Systems, Architectures and Processors, 2004
[25] Alizera Hodjat, Ingrid Verbauwhede,High-throughtput programmable cryptoprocessor, 2004
[26] HoWon Kim and Sunggu Lee, Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System, 2004
[27] Rahul Razdan and Michael D. Smith, A high-performance microarchitecture with hardware-programmable functional units, Proceedings of the 27th annual international symposium on Microarchitecture, 1994
[28] Bocchi, M. De Bartolomeis et all, R., A XiRisc-based SoC for embedded DSP applications, Custom Integrated Circuits Conference, 2004
[29] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, Ruby B. Lee and Niraj K. Jha, Impact of Configurability and Extensibility on IPSec Protocol Execution on Embedded Processors, VLSID '06: Proceedings of the 19th International Conference on VLSI Design, 2006
[30] Tensilica http://www.tensilica.com/
[31] Jeffrey M. Arnold, S5: The architecture and development flow of a software configurable processor, ICFPT 2005 : International Conference on Field-Programmable Technology, 2005
[32] Adronis Niyonkuru and Hans Christoph Zeidler, Designing a Runtime Reconfigurable Processor for General Purpose Applications, IEEE Computer Society, 2004
[33] Cowarehttp://www.coware.com
[34] Kimmo Puusaari, Application specific instruction set processor microarchitecture for UTMS-FDD cell search, International Symposium on System-on-Chip 2005
[35] Kimmo Jrvinen, Matti Tommiska and Jorma Skytt, A Compact MD5 and SHA-1 Co-Implementation Utilizing Algorithm Similarities, International Conference on Engineering of Reconfigurable Systems and Algorithms, 2005
[36] Guy Gogniat, Tilman Wolf, Wayne Burleson, Reconfigurable secutity primitive for embedded systems, International Symposium on System-on-Chip 2005, 2005
[37] G. Edward Suh et al, Design and Implementation of the AEGIS Single-Chip Secure Processor, 32nd Annual International Symposium on Computer Architecture, 2005
[38] Ricardo E; Gonzalez, stretch: a software configurable processor architecture, 2005