Trusted computing - A new challenge for embedded systems

Romain Vaslin, Guy Gogniat, Jean-Philippe Diguet
LESTER UBS/CNRS FRE 2734
Centre de recherche
56321 Lorient FRANCE
Email:firstname.lastname@univ.ubs.fr

Alain Pegatoquet
WTBU - CSSD
Texas Instruments
821, avenue Jack Kilby - B.P 5
06270 Villeneuve-Loubet Cedex FRANCE
Email: a-pegatoquet@ti.com

***Abstract*— Security issues become more and more important during the development of mobile devices. In this paper we propose a thorough overview of processor-based solutions to protect programs and data exchanges within embedded systems. A discussion about the limitations of existing solutions is provided and new directions are proposed.**

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 [1]. 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. The encryption of information is used for confidentiality. The most popular cipher algorithms are: RSA, ECC, AES, 3DES. 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 and SHA. In addition, non-repudiation, availability and authenticity are guaranteed by communication protocols like IPSec for example.
More and more security tasks are assigned to embedded systems. Thus, it becomes essential 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 level of system protection. It is essential that these solutions provide hardware architectures adapted to embedded systems to meet the tight constraints on memory size, performance and power consumption. In the following sections we propose a thorough overview of processor-based solutions to protect programs and data exchanges within embedded systems.

II. Secure architectures: state of the art

In order to fend off hardware and software attacks specific mechanisms have to be defined. All security solutions are built around assumptions concerning their potential threats. Generally, the secure zone is composed of the processor core and the ciphering and hashing dedicated blocks. Moreover side channel attacks are mostly not addressed.
In section II-A the studies focus on the protection of program memory and data memory. A monitor is used to protect the operating system (OS). Using an OS, there is a need to track if a task does not reach any secure information not belonging to it. 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. 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). 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. In section II-A we focus on the principles of the OS and not on the hardware engines to accelerate the computing of the cryptographic tasks. Section II-B details the hardware engines to efficiently implement encryption, decryption and hashing functions for embedded systems.

A. Program and data security software-based solutions

1) Trustzone [2]: 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. 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. 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.

2) XOM [3]: XOM is the acronym of eXecute Only Memory. XOM wishes to completely secure an architecture. 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 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 required for the asymmetric decryption is stored in the secure zone of the architecture. The signature result of the hash algorithm is compared to 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 programs which try 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).
All the protections added by the solution have a cost. The first one results from the implementation of XOM in an existing OS. A work is necessary on the kernel to add the instructions which handle 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%. This raise is due to the information added into the cache to secure the data and their associated identifier. 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 [4]: AEGIS is an OS solution like XOM. 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. As 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. 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 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 surface is one of them as it is increased by 1.9 [5]. The cpu core is the part which is the most concerned by this raising. Moreover, the logic needed to control the specific mechanisms contributes to the raise of the area. The global performances of the architecture depend

on specific parameters like the size 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. 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 ciphering algorithm (symmetric or asymmetric). Specific execution units are added into the datapath. [6] and [7] propose processors with instructions for symmetric ciphering algorithms. Specific instructions have been defined like logical operation (xor-add) or data permutation. For processors dedicated to asymmetric ciphering algorithms [8], specific instructions are defined. For instance to efficiently compute the modular exponentiation used in 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. [9] and [10] propose two hardware accelerators in order to speed up ciphering operations. Their architecture is fixed and controlled through configuration registers. The main feature of [9] is its ability to run several algorithms in parallel and to select the execution parameters associated to each security primitive. [10] is a configurable solution which allows the user to switch in different modes of the AES algorithm. In both cases the architecture is dedicated and optimized for an algorithm.

Another approach consists in 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 [11]. In [11] 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 [12]. 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. The results obtained with this solution are really interesting as for an implementation of DES algorithm, the speed up of the algorithm is about 13 times with the reconfigurable logic.

[13] have considered a similar approach for security applications. By exploiting the Xtensa architecture of Tensilica [14], 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 time. The analysis is performed during compilation and the reconfiguration is done at power-up. 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 [15] 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 to 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 cost and the power consumption and to provides efficient execution patterns to speed up the execution. In [16], it is shown that the reconfigurable coprocessor speeds up the architecture by 190 for a specific application (EEMBC).

A similar approach is proposed in [17] 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 [9] [10], leads to high performances but at the cost of power consumption which can be prohibitive in some cases.
In the case of configurable architecture [13] 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 supplier provides an efficient compiler which can identify and exploit specific instructions. For architectures like [15], 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.
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. 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 should be to define a (re)configurable architecture dealing with these algorithms. Each algorithm can be associated to a dedicated coprocessor with specific instructions.
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.

III. 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, the efficiency but also the programmability.

References

[1] D. Dagon, T. Martin, and T. Staner, Mobile Phones as Computing Devices: The Viruses are Coming!, IEEE Pervasive Computing, 2004
[2] ARM trustzone http://www.arm.com
[3] XOM project: http://www-vlsi.stanford.edu/ lie/xom.htm,
[4] AEGIS project: http://publications.csail.mit.edu/abstracts/abstracts05/suh/suh.html,
[5] G. Edward Suh et al, Design and Implementation of the AEGIS Single-Chip Secure Processor, 32nd Annual International Symposium on Computer Architecture, 2005
[6] Rainer Buchty, Nevin Heintze, and Dino Oliva, Cryptonite A Programmable Crypto Processor Architecture for High-Bandwidth Applications, 2004
[7] 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
[8] 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
[9] HoWon Kim and Sunggu Lee, Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System, 2004
[10] Alizera Hodjat, Ingrid Verbauwhede,High-throughtput programmable cryptoprocessor, 2004
[11] 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
[12] Bocchi, M. De Bartolomeis et all, R., A XiRisc-based SoC for embedded DSP applications, Custom Integrated Circuits Conference, 2004
[13] 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
[14] Tensilica http://www.tensilica.com/
[15] Jeffrey M. Arnold, S5: The architecture and development flow of a software configurable processor, ICFPT 2005 : International Conference on Field-Programmable Technology, 2005
[16] Ricardo E; Gonzalez, stretch: a software configurable processor architecture, 2005
[17] Adronis Niyonkuru and Hans Christoph Zeidler, Designing a Runtime Reconfigurable Processor for General Purpose Applications, IEEE Computer Society, 2004