Secure embedded system hardware design – A flexible security and trust enhanced approach

Apostolos P. Fournaris a,b,*, Nicolas Sklavos b

a Electrical and Computer Engineering Dept, University of Patras, Rio Campus, Patras 26500, Greece
b KNOSSOSnet Research Group, Informatics and Mass Media Dept, Technological Educational Institute of Patras, Greece

Abstract

Embedded systems have found a very strong foothold in global Information Technology (IT) market since they can provide very specialized and intricate functionality to a wide range of products. On the other hand, the migration of IT functionality to a plethora of new smart devices (like mobile phones, cars, aviation, game or households machines) has enabled the collection of a considerable number of data that can be characterized sensitive. Therefore, there is a need for protecting that data through IT security means. However, care usually deployed in hostile environments where they can be easily subject of physical attacks. In this paper, we provide an overview from ES hardware perspective of methods and mechanisms for providing strong security and trust. The various categories of physical attacks on security related embedded systems are presented along with countermeasures to thwart them and the importance of reconfigurable logic flexibility, adaptability and scalability along with trust protection mechanisms is highlighted. We adopt those mechanisms in order to propose a FPGA based embedded system hardware architecture capable of providing security and trust along with physical attack protection using trust zone separation. The benefits of such approach are discussed and a subsystem of the proposed architecture is implemented in FPGA technology as a proof of concept case study. From the performed analysis and implementation, it is concluded that flexibility, security and trust are fully realistic options for embedded system security enhancement.

1. Introduction

Modern trends in the information technology world dictate a highly interconnected environment where the user has constant access to information. Concepts like Future Internet and smart grids are rapidly becoming a reality while mobile device technology is considered already a well-established market bound to affect the life of millions of customers. This constant need for information access cannot be satisfied by traditional computing systems (Personal computers, servers, etc.). On the contrary, it points to specialized computing components meant for specific tasks in the new IT infused world. This can be achieved by embedding computing power to a wide variety of electric devices in an effort to provide them with intricate and smart functionality.

Embedded systems (ES) may be viewed as autonomous data computation, storage and communication modules, following the above approach, and they are rapidly expanding in a wide range of environments like industrial, military, crisis management, private and public (governmental, metropolitan) spaces, ranging from sensor networks, smart cards, mobile
devices to medical, avionics, automotive and smart/power grid controlling systems. Embedded systems of today handle an increasingly large amount of data from our personal, social, business, medical or civilian life including confidential information, security keys, private user habits etc. Such information are deemed sensitive and need to be protected. Embedded system security, considered in earlier years as an exotic ES feature is now becoming a necessity.

Infusing strong security to the embedded system paradigm is not an easy task. Embedded system specificities, both functional and non-functional, differ from traditional computation machines (desktop computers, servers, etc.) and can be very restrictive to the systems supported security. While ESs consist of a hardware, software and an environment similarly to traditional computing systems, due to their physical constrains, embedded system software is very closely interconnected to the hardware structure and the computation environment at hand [1]. The physical constrains of an embedded system are related to the available processing power and storage capacity, which in many cases can be very limited (sensors, RFIDs), but also on power dissipation or energy consumption which is very restricted, especially in battery operated systems.

Such constrains dictate specific requirements to the ES developed software as well as the hardware structure to manage it. Strong security, being a computational intensive set of operations, with non-negligible storage needs, can hardly fit to the above physical constrain framework.

Furthermore, many embedded systems are deployed in uncontrolled, stand-alone environments where are subject to possible intentional or unintentional hazards (e.g. in crisis management scenarios, military scenarios). This sort of environment highlights the need for embedded system robustness and introduces considerable risks for unintentional or intentional information leakage. From security perspective, the embedded system environment is viewed as hostile since it can lead to physical security attacks.

Finally, another important embedded system constrain with an impact on security, is production cost. Embedded systems are meant to be used in large volume applications (e.g. in automobiles, mobile devices, bank transactions) and need to be very cost-efficient. As such, designing, developing and implementing strong security features in off-the-shelf embedded system devices can considerably increase the overall device cost and expensive devices are usually not preferred by both, the manufacturer and the costumer. The fact that security operations are transparent to the end user and therefore cannot be easily valued, makes the above problem worse since customer does not easily consent paying the extra cost for strong security. All the above constrains, restrict the capabilities of an ES and highlight its significant exposure to security attacks. Such attacks are related to the systems hardware (physical attacks), software (logical attacks) and communication interface (network attacks) [2]. Physical attacks aim at extracting sensitive information from a hardware structure during security related hardware operation execution by measuring in a direct or indirect way hardware characteristics like power, electromagnetic emission, time delay or by invasive direct reading or modifying data in the ES hardware storage elements (memory, registers) or data flow. There is a wide variety of hardware attacks [3] in literature. They can be very successful in ESs since such systems, as already mentioned, are deployed on uncontrolled, hostile environments and can easily be subject of theft and manipulation.

Logical attacks are based on establishing malicious code (Trojans, worms, viruses etc.) within the software or firmware structures of the embedded Operating System (OS) in order to provide to an attacker unauthorized system privileges and capabilities (e.g. root access rights, data theft, hijacking) or cause the system to malfunction (e.g. energy depletion through Denial of Service, reduced functionality). Such attacks can be devastating in critical infrastructure embedded systems as recent security history has shown through the Stuxnet or Flame worm infections [4,5].

Since embedded systems are usually interconnected in an overall network, they are vulnerable to network related malicious attacks. Such attacks resample those on traditional computer systems, however, they can be more dangerous since the ES does not have strong protection mechanisms (firewalls, Intrusion Detection Systems, etc.) due to its constrained hardware resources.

As can be remarked from the above arguments, the hardware structure of the embedded system plays a very important role on the overall system security since it is very closely related to the software structure. Hardware constrains dominate the whole design as well as the ES implementation logic and can be viewed as the ES foundation of supporting strong security features. The ES security hardware structure can have impacts (positive or negative) on both the ES software and communication security. Many researchers [2,3,6] have suggested ways to enhance the overall ES security, firstly, by providing appropriate hardware structures that can act as dedicated cryptographic accelerators thus bypassing the ES generic processor restriction on computational power. Secondly by providing hardware security arbiters that measure software trust level (trust establishment approach) or perform dedicated monitoring of logic or network attacks. When designing such hardware structures, physical attacks must be taken into account and appropriate countermeasures must be realized.

In this paper, an overview of approaches and mechanisms is discussed for securing an embedded systems sensitive information and for providing attack resistance. Our analysis is focused on specifying and describing an ES hardware architecture structure capable of offering the above security and trust functionality while providing strong physical attack resistance. We analyze modern secure hardware design trends for embedded systems that can support security flexibility, adaptability and trust establishment. Based on these approaches we propose a security enhancing ES hardware architecture consisting of trusted and untrusted zones, offering flexibility through the use of reconfigurable logic hardware structure (FPGAs) and trust establishment through dedicated hardware modules. As a case study of the above proposal, an FPGA based trusted zone architecture is presented, its components are analyzed and its main functionality is described.

The rest of the paper is organized as follows. In Section 2 physical attacks on ES and their countermeasures are described. In Section 3, some of the most promising hardware approaches for embedded system security and trust are discussed. In
Section 4, a FPGA based security and trust hardware architecture is proposed and a case study, proof of concept, design is presented in Section 5. Finally, Section 6 concludes the paper.

2. Physical attacks on secure embedded systems

An embedded system operating in a hostile environment can be stolen and manipulated by an attacker in order to release sensitive information stored or processed in it. The system might be strongly protected from cryptanalytic attacks using provable secure cryptographic schemes but it can still be compromised when an adversary applies a physical attack, an attack on the implementation itself. Even if secret information is not extracted, the system may still be considered compromised when an attacker disrupts the hardware functionality or denies system services. Physical attacks can be discriminated into three major categories:

1. Invasive attacks. These attacks aim at physically disrupting the correct operation of an embedded chip. They involve removing chip packaging, micro probing of the chips activity (memory, registers, buses) and physically tampering – interfering with the chip functionality. Invasive attack techniques begin with chip depackaging, removal of on-chip protection layers (depassivation) and modification through probing tools. Through such tools one can modify the executed code or change values in Registers or simply observe the behavior of static/dynamic RAM blocks after power off since such units tend to remember values long after no power is applied to them. Invasive attacks are not easily mounted, require attackers with considerable on-chip expertise and expensive, specialized equipment like laser cutter microscope, micro probes, etc.

2. Semi invasive attacks. These attacks aim at observing the behavior of the embedded system chip after an attacker specialized triggering. Like invasive attacks, they require chip depackaging in order to get access to its surface. However, the passivation layer of the chip remains intact, as semi-invasive methods do not require depassivation or creating contacts to the internal lines. A fault attack is the most practical representative of semi-invasive attacks. A Fault attacks goal is to induce a fault in the computation flow of the chip during a cryptographic operation and observe the cryptographic result as the fault propagates. This result on an unprotected embedded chip can be used to deduce sensitive information. In fault attacks, faults are injected using power or clock glitches, extreme variations in temperature, UV radiation or even optical laser beam induction. Depending on the attackers equipment and expertise, the fault attacks are moderately difficult to mount.

3. Non-invasive attacks. Such attacks, also called side channel attacks (SCA), exploit an embedded systems hardware characteristics leakage (power dissipation, computation time, electromagnetic emission, etc.) to extract information about the processed data and use them to deduce sensitive information (cryptographic keys, messages, etc.). An attacker does not tamper with the chip in any way and need only make appropriate observations to mount a successful attack. Side channel attacks can be mounted very easily, cheaply, using a PC, a digital oscilloscope and some probes. Therefore, they can be mounted to an embedded system device by even the most inexperienced attacker. This ease of use, makes SCA very potent. Side channel attacks are usually categorized as follows:

   • Timing attacks. The attacker observes the computation time (clock cycle number) needed by the embedded system to provide a cryptographic result. Depending on the secret/private key that is employed, this time delay may vary and it can provide info about the keys Hamming weight. Furthermore, an attacker can start guessing key bits and observe which hypothesis results have the strongest correlation between the predicted results and the actual ones. Collecting enough samples the attacker can recover the whole key.

   • Power attacks. These attacks involve physical measurement of the power dissipation emitted from the chip during cryptographic operations. Simple power signal analysis can reveal what operation is performed in the chip (e.g. modular multiplication or squaring during an RSA operation) and since in most cases the operation is related to the secret/private keys this action can reveal the key itself. Even if the chip is protected against simple power analysis, it is not fully resistant against power attacks since differential power analysis can still lead to compromise. In differential power analysis, the attacker perform guesses about a secret/private key bit, collects the related to this hypothesis power signal and correlates it with the actual power signal. The strongest correlation between the hypothesis and the actual measurement is the correct guess. Taking enough power samples and correlations the secret/private key can be revealed [7–9].

   • Electromagnetic emission attacks. These attack use the electromagnetic radiation emitted from the embedded system chip for simple analysis or differential analysis in a similar way as power attacks. In electromagnetic attacks, specific chip areas can be targeted and no physical access to the chip is strictly required (they can be mounted from afar). On the other hand, the existence of physical noise, RF interference or measurement error limits the attacks effectiveness.

2.1. Countermeasures

Designing countermeasures for physical attacks is not an easy task. Each security and cryptographic algorithm has its own vulnerabilities when implemented in hardware and therefore a ubiquitous approach toward physical attack resistance is impossible. In general, two approaches for countermeasures are used in practice, algorithmic based countermeasures and
circuit based countermeasures. Algorithmic countermeasures aim at modifying the cryptographic algorithm and associated computer algebra operations so that when implemented in hardware, the associated leaked sensitive information will be minimized thus being of no use to an attacker. Such countermeasures are more focused to semi-invasive and non-invasive attacks. Circuit countermeasures are hardware structures added to a cryptographic algorithms hardware architecture, implementation or packaging, capable of detecting or thwarting a physical attack. Such countermeasures can be used to protect an embedded system against all kinds of physical attacks.

Invasive attack resistance is achieved by designing special structures during chip packaging and assembling in order to provide tamper evidence, detection and resistance. This may include mesh sensors implemented in the metal layer after packaging, consisting of serpentine patterns of ground and power lines that are short-circuited if attempts on depackaging or depassivation are done thus destroying the chip. Also, the on chip silicon layers can be designed in such a way that visual chip surface analysis through microscope is very difficult. Adding multiple silicon layers with metal layers in between is such a technique applied during chip fabrication. All the invasive attack countermeasures are circuit based countermeasures.

Semi-invasive attack resistance can be achieved by using some of the countermeasures for thwarting chip depackaging attempts (used in invasive attack resistance), however, usually such countermeasures are not enough or are too expensive. So, semi-invasive attack countermeasures are focused on detecting fault injection during cryptographic algorithm execution. One approach toward this end is to modify the cryptographic algorithm so as to support infective computation. The basic concept of infective computation is that any data error introduced by a fault will propagate throughout the cryptographic computation, thus ensuring that the final result appears random and useless to the attacker in the end. Another approach that can be combined with infective computation is the design of specialized fault detection units in the cryptographic algorithm hardware architecture capable of detecting single or multiple faults. Such units involve elegant circuit design as well as modifications in the cryptography algorithmic flow in order to include specific conditions between intermediate values that the fault detection unit must detect after the computations are concluded but before the cryptographic result is released. When faults are detected then a random number or zero value is released thus denying an attacker any useful information about secret/private keys.

There is a wide variety of non-invasive countermeasures depending on what side channel attack they thwart. The basic goal of all countermeasures is implementing the cryptographic architecture in such a way that the implementations characteristics like power consumption, timing or electromagnetic radiation, leaks as little as possible of the secret keys or data. This can be achieved either by scrambling the leakage signal in such a way that is unrelated to the secret information computed in the cryptographic unit or by minimizing the leakage as a whole so that it is very difficult for an attacker to use it for a side channel attack. The first approach is related to algorithmic countermeasures that aim at inserting randomization through the cryptographic algorithm computation flow by providing Boolean, or arithmetic (multiplicative or additive) masking of the secret information (multiplication or addition with a random number). These countermeasures, also known as blinding, is very useful against differential attacks since they aim at decorrelation of the secret data with the leakage itself. The second approach is related to algorithmic and mostly circuit countermeasures. Through special circuitry, like double rail technique, power rebalancing or additional dummy operations (redundancy), we aim at normalizing the leaked signals so that they remain unchanged during cryptographic operations. It should be mentioned however, that protection against side channel attacks is never expected to be absolute: a determined attacker with a vast amount of resources can eventually, given enough time and effort, compromise an implementation. The goal from cryptographic engineering perspective is to realize in the cryptographic accelerator enough side channel attack countermeasures so that an attack on the system becomes too expensive in effort or cost to be interesting [10].

3. Embedded system security design approaches

The hardware structure of a secure ES can be split into 4 main components, the input/output peripherals, the processor, the storage elements (memory) and the special purpose computing components, as depicted in Fig. 1. Input and output

![Fig. 1. Generic embedded system hardware architecture.](image-url)
peripherals are responsible for the ES communication with its external environment. They may include wired or wireless communication interfaces like UART, serial I2C, USB, SPI, JTAG, LPC, Ethernet, Bluetooth, 802.11, UMTS, GPRS/3G etc. that can be directly connected to sensors and A/D converter structures capable of translating analog signals of the external environment to digital data that the ES can process.

The processing unit constitutes the core of the ES. It is responsible for executing software code, performing all necessary data processing and controlling the overall ES behavior. There are several approaches on how to design ES processing units that employ single purpose processors, general purpose processors, application-specific instruction-set processors (ASIPs) or reconfigurable logic units. Single purpose processors provide very specialized functionality, servicing exclusively the ES applications. In this case, such functionality is hardcoded in the control logic of the processor and is not reprogrammable. This approach main benefit is high efficiency since the hardware structure of the processor is dedicated into performing specific operations. However, single purpose processors offer very little or no flexibility.

ES with general purposed processors are based on software microcode stored inside the processor memory for supporting ES functionality. While this approach offers high flexibility, it can be considered inefficient and not very secure unless special security countermeasures are present on the general purpose processor and its software.

A compromise between the above two approaches is the use of ASIPs to realize ES functionality. In this approach, a special instruction set processor is adopted in order to provide operations fitting ES applications. Example of such processors can be DSPs that have special instructions for signal processing operations. The ASIP approach disadvantages are related with the high design and implementation cost and the need for specialized compilers. However, ASIPs offer good flexibility and efficiency and constitute a good compromise between single purpose and general purposed processor approaches.

Finally, reconfigurable logic (FPGA) based approach offers considerable flexibility and scalability. As technology improves, FPGAs are becoming very efficient and are claiming a part of the ES market. This approach can work independently of other processing structure through the use of softcore processors (Microblaze [11], Nios II [12]) realized inside the FPGA structure or in accordance with microcontrollers supporting the hardware components implemented within the FPGA. In both cases, the potentials offered by the use of reconfigurable logic (FPGA) are many since the designer has great freedom in adjusting the hardware behavior of the system by uploading appropriate FPGA configuration files (bistream files) from non-volatile memory during design or run time. From security perspective, FPGA design suffers from physical attacks that can extend beyond those of other hardware approaches including FPGA configuration file manipulation and high exposure to power of electromagnetic leakage [13]. Memory units are also very important components of an ES. Typically, ES have volatile memories like SRAMs for temporal value storage and non-volatile memories, like Flash memories, for permanent data storage. Memory units are needed for storing the software code to be executed by the processor units and the data values that the ES handles. Memory is the target of many security attacks since it includes security protocol software code, secret keys, credentials and sensitive information in general.

3.1. Securing the embedded system

When enhancing the above ES hardware model with strong security features, we should take into account ES security specificities like efficiency, respect to ES constrains, physical attack protection and cost. The strong bond between ES hardware and ES software should always be acknowledged and retained in order to support all software security services and provide hardware protection mechanisms against software attacks. Therefore, special modules within the hardware structure should be constructed in order to achieve the above goals. Toward this end, the ES hardware structure is enhanced with special purpose hardware components that complement the main processor security functionality. Such components can be integrated directly to the processing units data path, thus operating as coprocessors, using the main processors registers and memory structure to provide security related functionality. This approach fits well with single purpose processor based ESs. Furthermore, security related components can also be integrated inside the processors structure providing a security related instruction set to be used by ES Operating System (ESOS) and ES applications. This approach fits well to the ASIP based ESs. Finally, security components can be added as peripherals to the ES main bus. The main processor inputs security related data on the bus and provides appropriate control signals to security components requesting security operations. Components in this approach act as security accelerators and fit well with special purpose processor and generic processor based ESs. The various types of security related components are presented in Table 1.

<table>
<thead>
<tr>
<th>Type of Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>The main component of ES.</td>
</tr>
<tr>
<td>Memory</td>
<td>Stores data and program code.</td>
</tr>
<tr>
<td>Security Component</td>
<td>Specialized components for security.</td>
</tr>
</tbody>
</table>

Based on the above components, the secure embedded system hardware structure can be divided into 2 different zones, the secure or trusted zone and the insecure zone. All security critical ES processing or data transactions are performed in the secure zone of the ES while all non-sensitive transactions are left to be performed in the insecure zone. The secure zone includes the processor unit, along with security related components, as those presented in Table 1, and it may also include specialized, secured, physically protected memory units for sensitive program or data (key) storage (RAM, FLASH memory). Depending on the ES functionality, the secure zone may include other structures like a secure OS kernel, or secure boot-loader. The insecure zone includes any other hardware structures that do not handle sensitive data. Such structures include Input/Output interfaces, Analog to Digital converters, the ES main memory and non-volatile memory.

Special care must be given to information passing from the ES secure zone to the insecure zone. Some of that information may not leave the secure zone at all, like the root keys characterizing the ES, while other information cannot leave the secure zone unencrypted. Strong encryption must be applied to the security related data stored or processed outside the secure zone, appropriate integrity credentials must be presented for inserting such data from the insecure zone back to
Countermeasures like tamper resistant coating, side channel attack resistance, hardware process monitoring or trust involving all ES layers from power-on, hardware initialization, boot loader, OS kernel to application layer. However, such strong security protection approach can have a negative impact both to ES power consumption, computation load and hardware resource cost. They are vulnerable to physical attacks. Examples of block ciphers are AES, 3DES, Blowfish. Lightweight block ciphers are also used in practice for ES, like DESL, DESXL, CURU-PIRA, HIGHT, PUFFIN, PRESENT or XTEA [14].

Stream ciphers: Stream ciphers operate serially by generating a stream of pseudorandom key bits, the key-stream (stream ciphers are also called pseudorandom number generators) from a set of parameters, the secret key, K, and the initialization vector, IV. The secret key cannot be used many times. Stream cipher algorithms must be very compact and lightweight. They are mildly vulnerable to physical attacks. There are many stream ciphers used in practice like ZUC, Snow3g, Trivium, E0, F-FCSR-H v2, Grain v1 or Mickey v2 [15].

Asymmetric key ciphers: In these algorithms the key for encryption (public key) is different from the key for decryption (private key). The private key is only known to the entity who issues it while the public key is known to everyone. Asymmetric key ciphers have very high computational cost and require extensive storage area because they use very big keys. Embedded systems, working under a constrained hardware framework, need dedicated, low power, lightweight hardware structures for asymmetric cryptography functionality. Such structures can be very vulnerable to physical attacks. Examples of Asymmetric Key ciphers are RSA, EIGamal or Elliptic Curve cryptographic schemes [16].

Hash function units: These algorithms can generate from a variable bit length value a constant bit length value. Knowledge of the outcome value is impossible to provide knowledge of the input value (one time pads). Hash functions can be used for providing integrity to the ES data. Examples of hash functions are SHA1, MDS, SHA256-512, SHA3 and WHIRLPOOL [16].

<table>
<thead>
<tr>
<th>Security component</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic accelerators</td>
<td>• Symmetric key block ciphers: In these algorithms, the encryption key can be calculated from the decryption key and vice versa. In most symmetric key algorithms, the encryption key and the decryption key are the same. This key is used for encrypting a plaintext into a ciphertext and vice versa. These algorithms the sender and receiver agree on a key before they can communicate securely. They are designed so as to have small computation and hardware resource cost. They are vulnerable to physical attacks. Examples of block ciphers are AES, 3DES, Blowfish. Lightweight block ciphers are also used in practice for ES, like DESL, DESXL, CURU-PIRA, HIGHT, PUFFIN, PRESENT or XTEA [14].</td>
</tr>
<tr>
<td>True Random Number Generator (TRNG) units</td>
<td>TRNG unit is responsible for generating truly random numbers for use in security protocols and cryptographic operations. True randomness is achieved by using unpredictable physical phenomena like quantum physics or sources of entropy (radioactive decay, thermal noise, shot noise)</td>
</tr>
<tr>
<td>Physical Unclonable Function (PUF) units</td>
<td>PUFs are functions embodied in a physical structure that is easy to evaluate but hard to predict acting as as hardware one way functions [17,18]. PUF evaluation is managed when matching a set of challenge-signals to some response of its specific hardware structure. Uniqueness is achieved through the PUF design that is based on physical process (e.g. the silicon chip manufacturing variations during fabrication) that cannot be replicated. They can be used for providing authentication, unclonability, key generation to the ES. PUFs need very few hardware gates to be implemented and they are very performance efficient, however, their responses are influenced by external parameters like temperature or power fluctuations so error correction algorithms must be applied to provide a reliable outcome</td>
</tr>
<tr>
<td>Process monitoring units</td>
<td>Such hardware structures aim at continually monitoring the instructions executed by the main ES processor and compare them with expected behavior of the program as derived from the execution code [19–22]. The processor sends a stream of information to the monitoring unit in order to provide such functionality. Using this approach, logic attacks can be contained or avoided since deviation from expected behavior indicates the presence of malicious code</td>
</tr>
<tr>
<td>Trust establishment units</td>
<td>An expansion of the monitoring units that secures the overall processing functionality of the ES including all hardware and software transactions, memory accesses, etc. Enables a trust zone inside the ES that can be used for security related activities. Such components while effectively protecting against logic attacks they can still be susceptible to physical attack</td>
</tr>
</tbody>
</table>

the secure one. Obviously, the secure zone must be protected against all possible attacks including physical and logical ones. Countermeasures like tamper resistant coating, side channel attack resistance, hardware process monitoring [20] or trust establishment units are usually present in the secure zone.

3.2. Trust establishment in embedded systems

Trust establishment can be a very effective way of infusing strong security and trust in an ES. It can be achieved by providing a fully protected, measured and controlled environment against security attacks through hardware means. Similar to the ES secure zone approach, trust establishment methodologies lead to trust zones inside the ES that provide strong security and trust. Trust zone establishment goes beyond process monitoring or bus encryption approaches. The trust zone security protection functionality is more extensive than that of the secure zone approach since it provides a ubiquitous path toward trust involving all ES layers from power-on, hardware initialization, boot loader, OS kernel to application layer. However, such strong security protection approach can have a negative impact both to ES power consumption, computation load and hardware resources. ARM in its latest processor families offers the TrustZone framework for ES security and trust. These processors, using TrustZone, are capable of splitting their functionality between a secure and non-secure operation zone, thus making possible the realization of a secure, trust anchor within its core. So, an ES developer can use the secure part of the processor by running a Trusted Execution Environment, separated from the main OS and protected from software/malware attacks in order to implement trusted applications. The TrustZone switch into secure mode through appropriate components provides hardware backed isolation from the insecure section. This security is achieved by partitioning all of the System on Chip hardware and software resources between the secure and non-secure zones of the processor. The TrustZone
framework inside the ARM-Cortex processor family, monitors operations of the OS, characterizes them as trusted or not and
allows their execution to the secure zone accordingly. TrustZone main functionality is related with access rights to sensitive
program code or data (e.g. keys, boot loader) inside the memory. However, it does not provide hardware structures for cryp-
tographic functions. These functions can be implemented in the software of the secure zone. Virtualization is also a technol-
ogy with potential uses in securing the embedded system world. By providing a software environment where not only
programs but also operating systems can run as if executed on hardware, virtualization achieves strong isolation. Thus, logi-
cal or network attacks on an application of an OS that is executed on top of a virtual machine (VM) can be contained there
and wont spread to the rest of the system. Fig. 2 provides example of such system.

The ES designer can create trusted areas of virtual machines running on a virtualization core on top of the ES hardware
and direct sensitive applications and data toward those virtual machines. Extending this logic, trusted ES OS can be run on
such VM and as long as access is controlled by a Trusted Computing Base (TCB) program on the processor, the OS remains
isolated and protected from the rest of the ES untrusted VMs. ES virtualization technology is still on an early stage, since
several practical implementation problems still exist (ES hardware constrains, ES real time behavior, scheduling, access con-
control rights) [24]. Currently, ES virtualization has been successful using microkernels like the L4 family [23–26].

Trusted Computing, supported by a strong consortium of companies and organizations (Trusted Computing Group, TCG)
[27], is also a very promising technology for ES strong security and trust. Trusted computing can be viewed as a collection
of technologies capable of constantly monitoring the behavior of a given computer system for indication of a possible compro-
mise. The core of this technology is embodied by a hardware chip, denoted as Trusted Platform Module (TPM). The TPM (TPM
v. 1.2) currently uses AES, RSA algorithms for encryption/decryption, authentication and privacy. TCG has established a work
that aims at applying the trust directives to the embedded system world. The trusted computing approach through the
TPM provides strong security services including hardware cryptographic acceleration, cloning protection, trusted embedded
software update, secure storage and authentication. Such functionality is supported through the TPM hardware structure
and the associated Trusted Software Stack (TSS) [28].

TPM trust measurement is done by establishing a root of trust for the system's behavior. TPM is activated from the ES
power on and measures the system from boot through a daisy chain process (chain of trust) [29]. After power on, the
TPM root of trust is the first to gain control of the system, then the boot loader is measured and if trusted, control is passed
to it. The chain of trust is established in a similar fashion through all the stages in the ES boot sequence. If the chain is not
broken in any stage of the boot sequence then the system can be trusted. The chain of trust functionality is implemented
through the TPM’s Platform Configuration Registers (PCR). Data provided by each measurement are always concatenated
to the value of an appropriate PCR that contains a hash value of a previous measurement. The result of this concatenation
is hashed and the outcome is stored in the same PCR (extend operation). A history of the extend operations for a specific
PCR value is kept outside the TPM. The sequence of extends on the history file, is compared to the current PCR value in
the TPM and must be the same for trust to be ensured. This chain of trust functionality can be reported to the external envi-
ronment as a proof of trust. The TPM based trusted computing can be used as a very sophisticated TCB for providing virtu-
alization [6,30].

3.3. Reconfigurable hardware for embedded system security

Reconfigurable hardware, like PLDs and FPGAs, can play an important role in the design of secure embedded systems.
FPGA technology is very mature and can offer important benefits for security enhancement:

Flexibility: The FPGA design being reconfigurable, enables the reprogramming of hardware cryptographic algorithms
either during design phase thus providing algorithm updates or at run time thus offering flexibility, adaptability. Attack pro-
tection countermeasures can also be updated thus retaining and enhancing a strong security level during the ES life-cycle.
Efficiency: While FPGAs are not as efficient (slower speed, higher chip covered area) as dedicated hardware chips (i.e. ASICs and ASIPs) their adaptability can lead to very closely matched designs to an ES security specifications. The architecture efficiency, by adapting through FPGA reprogramming the security parameters of the system to its specific needs, can be very high. While ASICs or ASIPs can change parameters through software means, the FPGA designer can reprogram the hardware itself which leads to much faster results. Similarly, hardware resources efficiency through FPGA can also be very high since instead of implementing all cryptographic operations in hardware like the ASIC approach, in FPGA design we just have to implement one such operation and through reprogramming, change the hardware structure according to security needs. Note also, that in FPGA design all security functions can be implemented in hardware in contrast to other approaches were dedicated, non-reconfigurable, hardware is present and many security functions are implemented as software programs on top of a special purpose or general processing unit. Under this light, FPGA security implementations can behave more efficiently than non-reconfigurable ones.

Cost: The development cost of FPGA implementations is considerably lower than that of ASIC implementations. The extensive tool-set for FPGA testing and verification is simpler and security related misses like physical attack vulnerabilities (fault attacks, side channel attacks) can be discovered and corrected easily compared to ASIC approaches.

However, ES secure FPGA implementations are subject to attacks related to their reconfigurability. Most FPGA based systems have some specific storage area (non-volatile memory) where FPGA configuration files (bitstream files) are kept. Such files are target of attacks since they define the hardware architecture and functionality realized in the FPGA. Goal of each attack is to read back the configuration file either using the JTAG port (read back attack) or physically attacking the non-volatile memory where the file is stored [13]. Obtaining the FPGA configuration file enables an attacker to clone the FPGA hardware implementation and use it in another device or reverse engineer the file in order to obtain the hardware data structure used by the manufacturer. Such attacks constitute a breach on the FPGA implementation IP rights but are also useful tools for strengthening physical attacks like template attacks or side channel attacks [6]. However, many FPGA manufacturers in their latest FPGA versions provided protection mechanisms for the FPGA configuration file, like configuration file encryption or on board configuration file secure storage space.

4. FPGA based security protected ES hardware architecture

Using the previous analysis, an ES hardware architecture, capable of supporting strong security and trust, can be derived. Such architecture follows the most promising approaches for security enhancement in an effort to satisfy the following basic directives:

- **Security processing:** The embedded system main processor does not always have the necessary processing power to run security demanding operations. Thus, special care must be paid by the systems designer to enhance systems processing power or provide alternative processing sources.
- **Embedded System constrains match:** The ES resources can be limited, thus, the ES hardware architecture must be able to match constrains in power dissipation, processing power and needed hardware resources.
- **Flexibility:** Due to ever evolving security status, the need for interoperability in different environment and the versatility of security objectives, requires that the embedded systems security characteristics must be very flexible and adaptable.
- **Resistance Against attacks:** Embedded system devices can be the target of compromise by potential attackers. Countermeasures must be built against physical and logical attacks.
- **Cost:** The embedded system security can strongly influence ES design cost. The highest the level of security instilled in the embedded system device, the highest is the overall cost of implementing such device. An appropriate, affordable, trade-off between security and cost must be made.

We propose an approach to meet the above requirements, which is based on trust establishment structures within the ES architecture as well as reconfigurable hardware (i.e. FPGA technology) for support of demanding cryptographic and security related operations as well as cost reduction. The proposed system is split into three zones of operation, the processor trust zone, the FPGA trust zone and the untrusted zone, as described in Fig. 3. The processor trust zone consists of a trusted ES kernel capable of supporting security critical operations. This zone consists of the ES processor core, which can be a generic or single purpose embedded processor, a secure register file and cache memory along with a SRAM module containing security related program code and data. The functionality of the above structures is kept secure through a dedicated trust establishment unit acting as a TCB, following a chain of trust approach. We envision such unit as a tiny TPM structure similar to the one proposed in [31] or a full embedded TPM as proposed in [32,27] depending on the ES specifications and constrains. The Secure boot-loader, acting as an additional protection mechanism, is responsible for providing malicious code free boot process and complements the work of the Trust establishment unit. The processor trust zone includes a tamper-evident, secure, Flash memory module where sensitive information can be stored like keys and credentials as well as FPGA configuration files that can be loaded at run time to the FPGA trust zone, thus, supplementing its cryptography and security functionality. The trust establishment unit contains appropriate hardware cryptographic operation components so as to support strong security functionality (encryption/decryption, integrity, authentication). All data entering or leaving the processor trust zone must be checked by the Trust establishment unit and categorized as critical (security related) or non-critical.
This categorization is related to the zone from/to data are inputted/outputted respectively. Data coming from the untrusted zone are considered non-critical while data from/to the FPGA trust zone are considered critical and they are encrypted and checked for integrity and trust conformity. Strong isolation mechanism, supported by Hardware virtualization, can be used to support the above functionality.

The FPGA trust zone adds flexibility to the ES security architecture. The cryptographic components realized in FPGA technology can be updated with the latest supported security algorithms and protocols in an effort to provide long term security. This approach offers all the FPGA security benefits described in Section 3.3 and avoids issues stemming from vulnerabilities related to the FPGA configuration file since this file is stored securely in the processor trust zone. The FPGA trust zone communicates securely with the processor trust zone, providing strong cryptographic operation functionality that cannot be performed in the processor trust zone. Such operations may include Symmetric and Asymmetric key cryptography, true random generation and hashing. Example operation can be pairing based cryptography, which is considered very computationally intensive; thus is not affordable from the main processor, but provides very intricate identification, authentication and privacy protocols (e.g. identity based cryptography, zero knowledge proof protocols).

Both processor and FPGA trust zones must be strongly protected against physical attacks by temper evident packaging and appropriate countermeasures. Especially, FPGA trust zone can be considered very secure due to its strong flexibility, enabling physical protection hardware countermeasure update and reconfigurability through new FPGA configuration files by JTAG port. We assume that the FPGA manufacturer provides appropriate JTAG protection mechanisms [11,12].

Finally, the untrusted zone handles all security unrelated ES operations and includes input/output interfaces as well as the main ES memory.

5. Case study: Asymmetric key cryptographic primitive embedded system design platform

As a case study of the proposed approach, we provide an implementation of the FPGA trust zone involving a microcontroller, communication interface and an asymmetric cryptographic primitive accelerator. Such a system constitutes a proof of concept for the FPGA trust zone since it satisfies this zones main goal of flexibility and computer intensive operation acceleration.

The proof of concept architecture is developed purely on Virtex 5 FPGA technology and consists of a Microblaze softcore processor and associated Input/output interface along with a custom design of a physical attack resistant RSA core accelerator. The FPGA trust zone proof of concept architecture is meant to operate in association with a TPM based processor trust zone structure and includes an appropriate protocol framework for security activities like access verification and session key generation. Such framework is integrated as a software code in the Microblaze softcore microcontroller. The proof of concept architecture also has a secure key storage mechanism in an external flash memory. Inside this memory, keys, credentials and trust quotas (TPM trust proofs for each sensitive data from the processor trust zone) are stored. Such information are accessed only through key release protocols based on trust verification. The overall architecture of the proof of concept system is presented in Fig. 4.

5.1. Asymmetric key cryptographic primitive accelerator

The designed cryptographic core in the architecture of Fig. 4 is a 1024 bit RSA accelerator with strong fault and side channel attack countermeasures. The architecture is based on work done in [33–35] and uses a modified version of square and...
always multiply approach along with message blinding to provide resistance against both Simple, Differential Power and Fault Attacks. Our goal is to support an one for all protection mechanism that avoids many known vulnerabilities generated by combining several physical attack countermeasures. The RSA architecture, presented in Fig. 5, consists of 4 parallel Montgomery modular multipliers capable of concurrently calculating products needed for one round of modular exponentiation (the core RSA crypto operation). The algorithm used in the proposed design has the following form [33]:

**Physical Attack Resistant RSA algorithm**

**Input:** $c, b, b^{-1}, e = (1, e_1, \ldots, e_0), M$

**Output:** $(s_0, s_1, s_2, s_4) = (b^{e \cdot c^{-1}} \cdot b^{2^{t-1}} \cdot c^{e+1} \mod M, b^{2^t} \cdot c^{2^t} \mod M, b^{e^{-1} \cdot c^{-1}} \mod M)$

Initialization: $T = R^2 \mod M, s_0 = s_1 = b_R = b \cdot R \mod M, s_3 = s_4 = s_5 = b_{R,1} = b^{-1} \cdot R \mod M$, where $R = 2^{n+2}$

**Precomputation:**

1. $T_R = T \cdot c \cdot R^{-1} \mod M$
2. $s_2 = b_R \cdot T_R \cdot R^{-1} \mod M$

**Main exponentiation:**

3. For $i = 0$ to $t - 1$
   (a) If $e_i = 1$ then
      i. $s_0 = s_0 \cdot s_2 \cdot R^{-1} \mod M$, 

---

Fig. 4. FPGA trust zone proof of concept architecture.

Fig. 5. Physical attack resistant RSA architecture design flow.
the proposed architecture of Fig. 3. Encrypted communication in the proof of concept architecture is implemented in software on the Microblaze processor. In a fact is verified, the needed key can leave the FPGA trust zone. This approach is similar to the ones described in [37–39]. The application requesting the key and submits it securely to the FPGA trust zone. The Microblaze processor in this zone along with the asymmetric cryptography accelerator is responsible for verifying that the submitted trust quota is the same as the one stored inside the secure flash memory that is associated with the requesting application and key. Only when this

\[ s_1 = s_2 \cdot r \mod m, \quad s_3 = s_2 \cdot r \mod m \]

\[ s_2 = s_2 \cdot r \mod m, \quad s_3 = s_2 \cdot r \mod m \]

\[ s_0 = s_0 \cdot b \cdot r \mod m, \quad s_1 = s_1 \cdot c \cdot r \mod m, \quad s_2 = s_2 \cdot 1 \cdot r \mod m, \quad s_4 = s_4 \cdot b \cdot r \mod m \]

If values of \( i, e \) are not modified and \( s_0 \cdot s_1 \cdot r^{-1} \mod M = s_2 \cdot 1 \cdot r^{-1} \mod M \) then return \( s_0 \cdot s_4 \mod M \) else return error

Note that \( b \) is a random number, \( c \) is the message to be encrypted and \( e \) is the RSA public or private key while \( M \) is the RSA public modulus.

Implementing the above algorithm of the proof of concept architecture on Xilinx Virtex 5 xupv5-1x110T development board resulted in a 1024 bit RSA hardware implementation needing 11,570 slices, capable of working at maximum frequency of 219 MHz and providing an RSA output at average, after 4.82 msec.

5.2. MicroBlaze processor and security protocol framework

The Microblaze processor having a generic processor structure can have great adaptability to a systems needed functionality and constrains. It is implemented inside the FPGA and its parameters can be changed during run time when a new FPGA configuration file is loaded. On this softcore processor, we have developed all necessary driver libraries so that the asymmetric crypto accelerator operations can be fully supported. The processor has also been provided with a testing mode, where controlling commands can be given from serial cable connected to a PC. This provided us with a toolset capable of validating the FPGA trust zone proof of concept functionality lacking an ES processor trust zone (this role can be played be a TPM enabled PC).

Currently, the FPGA trust zone proof of concept architecture through the Microblaze processor supports security protocols for:

1. Accessing FPGA trust zone where storage of sensitive information from the processor trust zone is done.
2. Establishing a session key for symmetric key encrypted communication between the processor and FPGA trust zones.

The above described FPGA trust zone proof of concept architecture, described in Fig. 4, was implemented in the Xilinx Virtex 5 xupv5-1x110T development board, it included a microblaze softcore processor and utilized the development board’s peripherals for communication as well as the flash memory for secure storage. The remaining system of Fig. 3 including the processor trust zone, was simulated by a developed PC software verification and testing Python based program connected with the FPGA board through UART serial interface. The implementation results provided by the Xilinx EDK XPS toolbox are presented in Table 2.

The secure flash memory is considered tamper resistant and includes keys/credentials from applications running on the processor trust zone along with TPM log files, denoted as quotas, from existing trusted states of the ES main processor applications related to those keys. To access one such key, the processor of the processor trust zone, provides a new trust quota for the application requesting the key and submits it securely to the FPGA trust zone. The Microblaze processor in this zone along with the asymmetric cryptography accelerator is responsible for verifying that the submitted trust quota is the same as the one stored inside the secure flash memory that is associated with the requesting application and key. Only when this fact is verified, the needed key can leave the FPGA trust zone. This approach is similar to the ones described in [37–39]. The same approach toward accessing the flash memory stored keys can be followed for FPGA reconfigurable files to be loaded in the FPGA trust zone (thus changing the hardware structure of that zone).

Session key agreement follows the Needham–Schroeder–Lowe protocol [36]. The symmetric key algorithm to be used for encrypted communication in the proof of concept architecture is implemented in software on the Microblaze processor. In a real system, a symmetric key hardware accelerator core could also be implemented in the FPGA trust zone as described in the proposed architecture of Fig. 3.

<table>
<thead>
<tr>
<th>FPGA trust zone proof of concept implementation results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA</td>
</tr>
<tr>
<td>LUT number</td>
</tr>
<tr>
<td>Flip flop number</td>
</tr>
<tr>
<td>operation frequency</td>
</tr>
<tr>
<td>Protocol operation time delay</td>
</tr>
<tr>
<td>RSA encryption</td>
</tr>
<tr>
<td>Session key establishment [36]</td>
</tr>
<tr>
<td>(4 RSA op with no comm delay)</td>
</tr>
</tbody>
</table>
6. Conclusions

In this paper, an approach toward designing a secure and trusted ES hardware architecture was proposed following FPGA reconfigurability and trusted computing principles. Our goal was to review promising mechanisms towards ES security enhancement and adopt them along with strong physical protection measures in our proposed design. We implemented a subsystem of the proposed approach in FPGA technology as a proof of concept for correct functionality. Concluding the papers analysis it can be remarked that as ES are bound to become part of our lives, the need for strong security and trust will become so demanding that trust establishment and flexibility approaches, being viable security enhancement methodologies as suggested by this papers work, will be widely adopted in the ES IT market in the near future.

References


Nicolas Sklavos received the PhD degree in 2004 from the Electrical and Computer Engineering Department, School of Engineering, University of Patras, Greece. Currently, he is an Assistant Professor, with the Computer and Informatics Engineering Department, Technological Educational Institute Western Greece. He is Director of KNOSSOSnet Research Group: Security and Systems Design Group. His research interests include Cryptographic Engineering, Security, System-on-Chip and FPGA System Design.