Article

Anomaly Detection Trusted Hardware Sensors for Critical Infrastructure Legacy Devices

Apostolos P. Fournaris [1,‡*], Charalambos Dimopoulos[2,‡], Konstantinos Lampropoulos [2] Odysseas Koufopavlou [2]

[1] Industrial Systems Institute, R.C. ATHENA, Patras Science Park, Greece.
[2] Electrical and Computer Engineering Dpt. University of Patras, Rion Campus, Greece.
* Correspondence: fournaris@isi.gr
‡ These authors contributed equally to this work.

Abstract: Critical Infrastructures and associated real time Informational systems need some security protection mechanisms that will be able to detect and respond to possible attacks. For this reason, Anomaly Detection Systems (ADS), as part of a Security Information and Event Management (SIEM) system, are needed for constantly monitoring and identifying potential threats inside an Information Technology (IT) System. Typically, ADS collect information from various sources within a CI system using security sensors or agents and correlate those information so as to identify anomaly events. Such sensors though in a CI setting (factories, power plants, remote locations) may be placed in open areas and left unattended thus becoming targets themselves of security attacks. They can be tampering and malicious manipulated so that they provide false data that may lead an ADS or SIEM system to falsely comprehend the CI current security status. In this paper, we describe existing approaches on security monitoring in critical infrastructures and focus on how to collect security sensor - agent information in a secure and trusted way. We then introduce the concept of hardware assisted security sensor information collection that improve the level if trust (by hardware means) and also increase the responsiveness of the sensor. Thus, we propose a Hardware Security Token (HST) that when connected to a CI Host, it acts as a secure anchor for security agent information collection. We describe the HST functionality, its association with a host device, its expected role and its log monitoring mechanism. We also provide information on how security can be established between the Host device and the HST.Then, we introduce and describe the necessary Host components that need to be established in order to guarantee a high security level and correct HST functionality. We, also provide a realization-implementation of the HST overall concept in a FPGA SoC evaluation board and describe how the HST implementation can controlled. Finally, we provide indicative use case scenarios of how the HST can be used in practice to provide a variety of different security services beyond acting as a secure ADS sensor.

Keywords: security; hardware design; trust; cryptography; Anomaly Detection

---

1. Introduction

The past twenty years, the majority of the world's Critical Infrastructures (CIs) adopted many of the recent advances of the Information and Communication Technologies (ICT) field, to become more flexible and cost effective. This adaptation however, was rather hasty and without thorough evaluation of its impact on security. In particular, CIs have been installing numerous new devices with advanced computation power and connectivity capabilities, exposing their once closed and isolated systems to new types of threats and attacks. In 2013 the "SCADA StrangeLove team" demonstrated how to get full control of various industrial infrastructures including energy, oil and gas, chemical and transportation CIs, claiming to have found more than 60,000 exposed control systems online. Since then many incidents of cyber-attacks have been reported in airports and airlines, health insurance companies etc. There is an emerging need to fortify CIs since the impact of a major attack may have severe results in many levels (economy, public safety etc.) [1]

Protecting a CI system is a highly complicated task which requires the collaboration of multiple diverse security systems, technologies and mechanisms. Apart from the systems' security, CI protection includes training for the people, Private-Public Partnerships, Security Assessments, Vulnerability Analyses etc. At the same time, solutions like Intrusion Detection or Anomaly Detection systems, Malware, Ransomware and Antivirus tools, DDoS protection, Endpoint protection, Hardware protection for CI devices, Authentication and Authorization mechanisms etc., are only few of the technical tools required to fortify such complex environments. These tools are often integrated in a UTM (Unified Threat Management system) or a SIEM system. A SIEM system is different from UTM in the sense that it does not exactly integrates the individual security components, but instead collects information (e.g. logs, reports, events etc) from them and combines it with data from other sources, trying to "assemble the puzzle" and identify possible security risks.

Within this extensive area of security solutions, this work focuses on a very specific aspect of CI protection, the design of trusted sensors for Anomaly Detection Systems (ADS) and SIEMs. An ADS is a cybersecurity tool which extends the functionality of an Intrusion Detection System (IDS) and apart from monitoring very specific, predetermined network metrics, also collects information from other kinds of sources to estimate an IT system's cyber-security status. In CI systems these sources can be distributed sensors that generate logs which are then transmitted to a centralized ADS or SIEM analyzer. Obviously, the success rate of an ADS' detection process (small false positives or negatives number) relies on the ADS analyzer algorithm (nowadays based on machine learning) but also on the accuracy of the collected data. It is clear that maliciously manipulated sensors' data will lead an ADS in producing false results and keep the ADS user ignorant or falsely alert on possible cyber-security problems.

In this paper, we review the option of using hardware means in order to secure sensors' ADS/SIEM transmitted data, instilling trust in the overall process. Also, extending the work in [1], we propose a Hardware Security Token to be physically connected to legacy CI devices and act as a trusted ADS sensor for failed access attempts as well as a mechanism for provide authentication and integrity to sensor's collected data. In the paper, we analyze the HST architecture and approach and we describe how it can achieve a level of trust in the associated Host device using an appropriate security protocol. We also describe the main HST functionality achievable through the use of a dedicated Host software program for accessing the HST as well as the HST log reporting mechanism on the ADS monitoring system. Then we describe a realization-implementation of the HST using an FPGA SoC evaluation board and we show how the HST services can be accessed. Finally, we provide some use case scenarios where the HST functionality could be used. The rest of the paper is organized as follows. In Section 2, an overview of a CI ADS sensors is made, security issues that may arise are described and a relevant threat model is presented. In Section 3, mechanisms to create trusted ADS sensors are described and in Section 4 the hardware assisted sensor approach, architecture and functionality is proposed. Section 5 provides a realization of the HST along with use case scenarios of its usage and section 6 concludes the paper.

2. Critical Infrastructure Security Monitoring System Anomaly Detection Sensors

Considering the security threats and challenges that many critical infrastructures have it becomes obvious that there is a considerable need to continuously monitor such infrastructures during their regular operation for security anomalies that can be linked to some security attack. Typical IT systems have a series of well-developed tools that, using a wide range of technologies and methods, can detect, respond and mitigate security attacks. The generic category of run-time monitoring systems may comprise of various components like intrusion detection systems (IDS), zero-vulnerability malware detectors and anomaly detectors that are all interconnected under a security information and event management (SIEM) system. The SIEM is usually responsible for the correlation between various events and logs to extract security alerts and make attack mitigation suggestions. However, CI SIEM runtime security monitoring must consider the CI specificities that differ from those of a typical IT system.

CI systems (CIS) have a close association with the physical world (they monitor and respond to physical processes) thus they constitute an ideal realization of cyber-physical systems (or system of systems) and should be approached in that way in terms of security. According to [2] there are four basic characteristics that distinguish a CIS from typical IT systems in term of runtime security intrusion detection. Due to their connection between the cyber and the physical world, the CIS devices measure physical phenomena and perform physical processes that are governed by the laws of physics. Thus, a CIS security monitoring system must perform physical process monitoring. Furthermore, typical CISs use many OT components thus they are highly focused on automation and time driven processes that realize closed control loops operating with no human intervention (and its associated unpredictability). This kind of behavior focused on Machine to Machine communications, increases the regularity and predictability of the CIS activities and make them attractive to attackers. Thus, the CIS security monitoring system should be able to monitor regularly closed control loops. Thirdly, the attack surface of a CIS is considerably broader than that of an IT system. CISs consist of many heterogeneous subsystems, including IT and OT devices. They follow a broad range of different, not IT related, control protocols like ISA 100, Modbus, CAN etc. Some of these devices and protocols have proprietary software or standards that may make IT countermeasures unfitting. This reason along with the fact that a successful CIS attack has high impact and thus high payoff, attracts very skilled attackers that can mount very sophisticated attacks on CPSs and CISs [8]. Such attacks are usually very hard to discover and document. Attackers exploit CIS zero-day vulnerabilities which would render many IT security monitoring toolsets useless (eg. knowledge-based toolsets [2]).

Lastly, many CIS consist of legacy hardware that is difficult to modify or physical access. Such components may be partially analog, have very limited installed software resources and can be dictated by physical processes. The biggest challenge in such legacy devices is how to install security monitoring sensors on then and how to predict/model their behavior correctly in order to detect possible anomalies. Since legacy devices do not have many computational resources, it becomes hard for the monitoring system to retain its real-time responsiveness when collecting security metrics from them.

Runtime Security monitoring in the CIS domain, considering the above specificities, can take various forms. Monitoring relies on two core functions, the collection of data from various CIS sources and the analysis of data in a dedicated runtime security monitoring subsystem. To achieve appropriate data collection, the security monitoring system must deploy security agent software on the monitored CIS devices or introduce virtual entities (Virtual Machines) for data collection [29] within the CIS infrastructure. All collected data are analyzed in the CIS runtime security monitoring system that uses data mining, machine learning, pattern recognition or statistical data analysis to extract metrics on security issues that may take place inside the CIS at runtime. Such issues may be possible incidents, threats that can be binary characterized as bad/good or continuously characterized by a specific significance grade. The performance of the security monitoring system is measured by the False Positive Rate (FPR), the False Negative Rate (FNR) and the True Positive Rate (TPR). The system is also measured in term of incident detection latency and consumed resources number, computational overhead, excessive network traffic and power consumption [2].

Typically, SIEM and ADS applications can be considered the most integral component of a CIS security monitoring system. They rely on a broad network of remote and local sensor entities that are capable of collecting specific metrics on the health of a computing system and its network. Such sensors are programs like port scanners, honeypot entities, package analyzers as well as antimalware or antivirus software. Also, a very useful source of ADS input data is an Operating System's system log manager that is capable of collecting various OS activities including successful or failed authentication and authorization. All the above sensors track activities, log them and provide to the ADS appropriate events when some specific prerequisites (usually defined by the security administrator) are evident. An ADS can collect such events in real time or near real time, providing to the security administrators with visual information regarding the overall system's security status and informing them of possible abnormalities that may be security breeches. A typical ADS sensor would be a software program that is installed on a host machine.

As mentioned CIS have considerable heterogeneity that is reflected on the diversity of sensors needed to collect and log information for a CI ADS. This diversity, the critical nature of the overall system and the fact that CI devices can be left unattended on CI premises (factories, power plants, remote locations) significantly increases the possibility of device tampering and malicious manipulation. Such a compromise can hardly be noticed by the device's software ADS sensor (the attack can be focused on the hardware level) and may lead to data manipulation on the ADS sensor's logging mechanism thus providing the ADS with fabricated data to be used in order to create false or suppress legitimate anomaly events on the security monitoring mechanism.

2.1. Threat model

There exist only few works that attempt to ensure trust in the information collection mechanism from a network's end points of a CIS security monitoring and ADS system [3][4][5]. Some approaches rely on securing the communication channel between end device and ADS/IDS monitoring system [4] or rely on securing the end node itself by ad-hoc, trusted computing based, mechanisms [3]. With out loss of generality we can assume that the main threats on the end point security monitoring sensors can be associated with attacks on a CIS end node that disrupt the sensor logging mechanism. This disruption can be achieved by manipulation of the communication channel between the device's sensor and the remote ADS/security monitor leading to integrity or authenticity threats, or can be achieved by hijacking the CIS node itself. In the second case, our threat model also considers that it is realistic for an attacker to gain access to the CIS nodes physical storage disk and not have full control of the node's memory (the attacker does not have root access to the CIS node's operating system). We also consider in our threat model, threats related to physical attacks using some invasive (tampering), semi invasive (fault injection) or non invasive attack (side channel analysis) since we can assume that CIS end nodes may be left unattended in hostile environments [6] [7]. We also consider as active threats, failed identification and authentication, failed access attempts on the node or the sensor itself. Such failures should be logged and be send in a trusted/secure way to the anomaly detection security monitor system for analysis.

3. Introducing Trust on software sensors

A well acclaimed approach to instill a level of trust on a computing system is to introduce in its architecture a trusted computing base (TCB). A TBC acts as a root of trust, a point of reference for the overall system that can be considered trusted. The dominant approach to achieve that is to include a hardware component (a security token) in a device's architecture capable of handling security services in a secure environment. This secure computation environment (TCB) supported by appropriate software can be used for security critical operations and can be considered hard to tamper [8]. There are several approaches on how to provide such a hardware root of trust mechanism both industrial and research based. Among the most important ones are the specifications provided by Trusted Computing Group (TCG) aiming to enforce trust on a system by prohibiting the execution of malicious code, by protecting sensitive data (mainly private keys), and by attesting the system's trust level to other entities. This is achieved by a constant evaluation of the computer system's security from boot time. Since software is impossible to remain un-tampered in order to form a TCB, TCG solution is based on hardware protection mechanisms along with software tools to establish trust. For this reason, TCG specifies a Hardware Security Module, the Trusted Platform Module (TPM), that is capable of acting as trust anchor within a computer system [9] [10].

3.1. Using Trusted Platform Modules

Following the above concept, a CI device can be considered trusted if it includes a fully installed TPM chip and has the necessary software stack introduced in its OS kernel to support the TCG trusted computing functionality. A TCG Trusted computing enabled device can be used in order to create a secure environment for executing an ADS sensor's software code [3]. While TPMs have a well-established market share in Personal computers they are still not broadly used in the cyber-physical systems and CI domain. Most embedded system devices as well as similar CI control elements (e.g. Programmable Logic controllers, PLCs) do not implement TPM chips. Although, TCG suggests a mechanism for trust for those devices (typically Low Energy and Low resources devices) without a TPM, denoted as Device Identifier Composition Engine (DICE), such mechanism far from been adopted in CI system OT and IT end nodes that are not PCs.

3.2. Using Virtual Environments

Hardware virtualization is a different technique that can be used in order to instill trust on a CI device. In this approach, an isolation mechanism generates isolated execution environments, some of which can be considered trusted. A system security designer can create trusted areas of virtual machines (VMs) running on virtualized hardware, and direct sensitive applications and data toward those VMs. Extending this logic, trusted OS can run on such a 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 system's untrusted VMs [8]. In such case, the employed TCB is a hypervisor structure that is set on top of the device hardware. This solution, while very promising has several practical implementation problems like hardware constrains, system real time behavior, scheduling and access control rights. However, virtualization is currently possible through the TCG's TPM and associated Trust mechanisms. Also, isolated environments using hardware virtualization (virtualization that is assisted through a processor Instruction Set) is currently been used (usually in association with TPMs) in Intel and AMD based systems [11]. However, there is no hardware virtualization in embedded system devices. At best, such devices can rely on the ARM's Trustzone technology [12] which, although not a hypervisor approach, enables isolation between trusted and not trusted execution (trust zone versus normal zone).

4. Proposed approach for Legacy Systems

While the newest IT based CI devices may have some mechanism of instilling trust, typical CI control devices that constitute the backbone of a CI control loop, still have legacy processing units that are not created for security but rather for safety and high, real time, responsiveness. Security Monitoring loggers installed on such devices need to rely on an execution environment that is capable of supporting the ADS sensors' functionality and that is protected from malicious entities. To achieve high security in legacy devices, it has been proposed in several works to introduce external security tokens that can be considered trusted [13] [8] [14] [15]. Having that in mind, extending the work in [1], we propose a Hardware Security Token (HST) that could be used as an external security element on legacy devices in order to instill a level of trust on collected ADS sensor logs and provide a series of security services to an associated host device and user. In the following subsections we extend, expand and analyze the HST architecture, functionality and services thus structuring a complete solution for CI legacy device security protection.

4.1. HST architecture

The HST is a synchronous System on Chip (SoC) device based on an ARM microprocessor with TrustZone support (e.g ARM Cortex A processor class) that is connected through an AMBA

AXI bus to a series of cryptographic accelerator peripheral IP cores and storage elements like RAM, ROM and NVRAM memory modules. The cryptography accelerator peripherals act as a security element of the ARM Trustzone enabled processor and consist of an RSA signature unit, an Elliptic Curve (EC) Point Operation unit (ECPO), a SHA256 hash function unit, as well as a symmetric key encryption/decryption and key generation unit (using the AES algorithm), following an architecture similar to the one presented in [14]. All the HST IP cores are protected against semi-invasive and non invasive attacks [16] [8] [17]. The HST, using the above cryptographic peripherals, is capable of generating and verifying digital signatures and certificates, performing key agreement schemes like Elliptic Curve Diffie Hellman Ephemeral (ECDHE) protocol or Needham-Schroeder-Lowe protocol as well as AES based encryption/decryption (AES-CBC, AES-CCM) and authenticated message integrity schemes (HMAC). Also, the HST has a series of Input/Output Interfaces including CAN bus, USB and Ethernet. The outline of the Hardware-Software hybrid architecture is presented in Figure 1. As can be seen in the above figure, all SoC components are interconnected in the Central Interconnect bus. Apart from hardware IPs the HST has a software stack that is capable of controling and coordicating all hardware assisted security operations (using customized Cryptographic IP drivers and cryptography libraries) as well as all communication (throug a serial concole interface) with an HST host machine. Finally, in the Figure 1, special mention should be made on the non volatile RAM unit which is realized as a QSPI Flash memory. This memory acts as storage space for all cryptographic, sensitive, information like public private keys, symmetric keys, HST states, users etc.

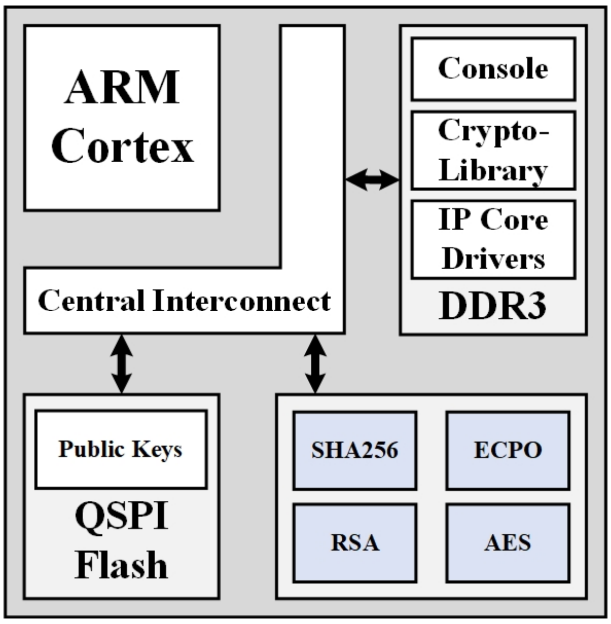

Figure 1. Hardware Security Module Architecture

The AMBA AXI (Central Interconnect) bus provides access to the cryptographic accelerator peripheral IP cores using the software stack. This stack implements a software component that is executed in the ARM cortex A trusted environment. The software component also handles the communication between the HST and the Host. During its operation, it polls for an input command given by the Host to the HST and collects all the necessary data each specific command requires. The required drivers that enable the stable operation of each IP core are included in this software component. Thus, the input data that have been collected during the command issuance are being propagated to the corresponding cryptographic peripheral for the output result to be calculated. Once this process is completed, our custom Crypto-Library is able to correctly perform a plethora of security protocols and algorithms that inherently depend on the operations our IP Cores provide. Protocols or algorithms that the Crypto-Library features are authenticated message integrity (HMAC), certificate generation and verification, digital signature schemes (ECDSA), as well as many utility operations (key generation, key validation, communication with Flash storage etc). The HST outputs to the Host through a secure channel the correct output data of the corresponding command.

The NVRAM (flash memory) module embedded on the Zynq 7000 series FPGA board can support the validity and functionality of the HST operations in a wide range of various use cases by offering a secure, self-contained and HST controllable storage area where sensitive information can be saved. A typical configuration of an HST's flash memory contents can be viewed in Figure 2.

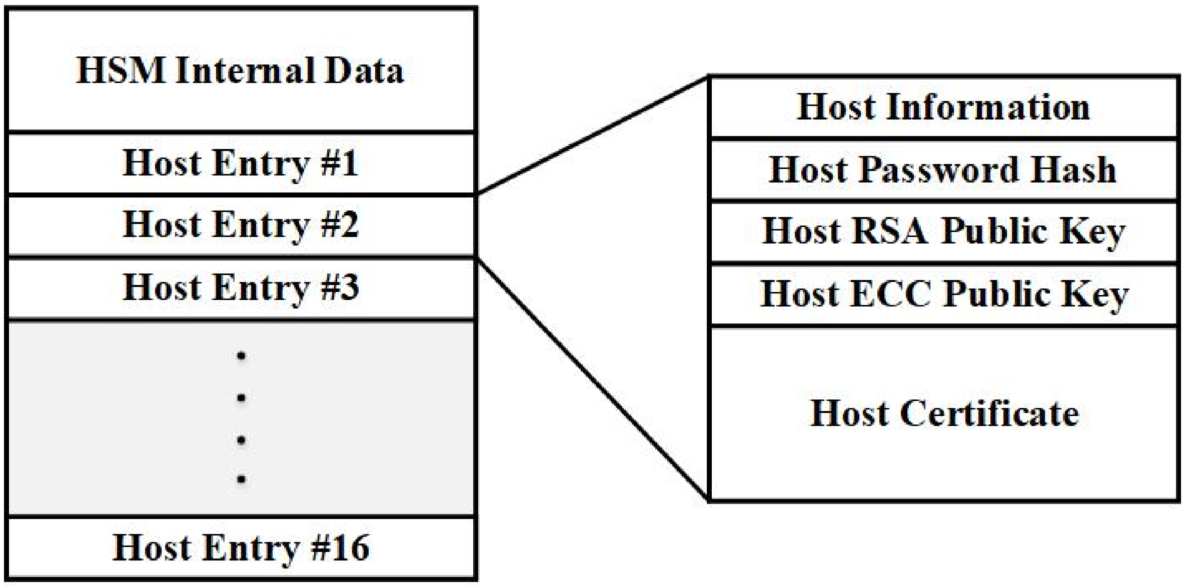

Figure 2. HSM NAND Flash memory contents

First and foremost, stored in the flash memory are all the HST specific information, including the HST ID, status and the highly sensitive private and public RSA and ECC key pairs. The remaining available storage can be utilized in order to store multiple Host Entries, each containing all the necessary information of the corresponding Host. More specifically, a Host Entry consists of basic Host information, including Host ID, Host type, Host status and a copy of the Password Hash that has been generated during the Host initialization phase. Additionally, each Host's RSA and ECC public keys are securely stored in the flash memory, along with a certificate that verifies the aforementioned keys (usually an ecdsa-with-sha256 based X-509 certificate due to storage limitations).

Apart from the above mentioned hardware structure, the HST has a dedicated software execution core that is retrieved from the HST flash memory and is loaded in the ARM processor RAM. This software core has dedicated components for the HST communication with the external world. More specifically the software core enables the HST, extending the functionality described in [14] and [1], to connected through usb cable to a Host device and through Ethernet to an IP network. The usb serial communication channel serves as a secure mean of communication between the Host device and the HST while the Ethernet based IP communication channel serves as a mean of communication between the HST and a remote ADS security monitor and analyzer. Apart from that, the HST software core is responsible for interfacing and usage of the hardware acceleration cryptography IP Cores (for the computationally demanding cryptography operations) using dedicated IP Core drivers. The HST software core also implements lightweight security operation that do not need hardware acceleration as well as security operations that during computations need some dedicated Hardware IP core output. Finally, the HST software core, implements the HST API that the HST uses in order to communicate with the associated Host device. The HST software component is been used in order to achieve two main operations, the Host to HST associated functionality and the HST logging mechanism.

4.2. Host to HST functionality

The HST can be used in order to identify and authenticate a host device, to collect a series of log entries from the host devices and transmit them through a secure channel to the ADS monitoring system. Apart from that, the HST is capable of providing individually security services to the host devices like certificate generation/verification, digital signatures, key agreement and secure channel establishment. Apart from the actual HST, the above functionality is possible by establishing a dedicated software component (HST/Host software component) on the host device that will act as a proxy between the Host and the HST. The HST/Host software component is acting on an untrusted environment in Host device, so we assume that it should not store sensitive information in the host device in a non secure way. This component is also responsible for the authentication of both the host user and the host machine to the overall ADS monitoring system. It also generates appropriate log entries using the syslog protocol and secures those entries using the HST. It does not solely rely on the Linux OS syslog mechanism but it also has a dedicated syslog client embedded in its structure in order to minimize a potential attacker's involvement in the logging approach. Finally, the HST/Host software component can relay to the HST commands and messages that are issued by the Host user using a dedicated HST Command Line Interface (HST CLI) as well as execute HST CLI scripts.

To achieve a secure use of the HST/Host software component and its access to the HST, an initialization phase needs to be occurred before HST deployment. In this phase, a trusted entity (a trusted host) must register to an HST the host device (through a device ID) that is associated with this HST and the user credential for this host device (through a password). Then the HST in its secure core will generate a salt (a random value) for this password and associate them using a Key Derivation Function (KDF) to create a symmetric cryptography key $Q = Q_{init}$. This salt along with a hash of the provided password are stored in the HST secure memory. Also, the HST will generate an Asymmetric cryptography key pair (public and private key) along with its certificate and will encrypt them using Q. This encrypted key pair/certificate will be send to the trusted entity. The trusted entity will finalize the initialization phase by sending to the Host, associated with this HST, this encrypted result. The Host can store this information in its storage areas (a hard drive disk or flash drive). An attacker that intercepts, copies and analyzes these files will fail to retrieve the key pair since he will not know the password nor the salt. Those two information are not stored in the Host machine, only in the associated HST. Only a user knowing the password and entering it to the host device connected to the HST associated with this device (where the salt is stored) can access the keys and eventually use the HST services (this acts as a two factor authentication). Failure to provide the appropriate password will generate a log entry (an abnormal event) that will be transmitted through the HST Ethernet dedicated channel to the ADS (using the HST logging mechanism). Taking into account that the HST is considered trusted, the ADS security monitor can trust that the HST sensor's collected input is not tampered. To achieve that, the entry is digitally signed with the specific HST's Asymmetric Cryptography key. We assume that the ADS has knowledge of all the HST sensors Asymmetric Cryptography public keys and their associated certificates.

After the Initialization phase, the HST-Host system is deployed in a CI system. The full association between a CI Host Device and its HST happens when, after they are connected through usb, they initiate a secure session.

The proposed key agreement scheme for such a session is presented in Figure 3. In this protocol, the two factor authentication of user and device is achieved along with an ECDHE for producing the (AES) session key in order to secure communication between the Host and the HST. Initially, the Host user must provide the registered password to the Host device. The password is not stored anywhere (apart from the Host's memory). The Host then requests the salt in order to decrypt the Host's/User's Certificate-key pair that are stored in the Host disk. For replay attack protection the HST generates a nonce (a random number) and using the user password's hash value as a key, it encrypts the nonce, the HST stored salt (salt1) and a newly generated salt value (salt2) using a Symmetric Key encryption algorithm ($E_K()$: AES) and transmits the result to the Host. After using the salt value, to retain forward

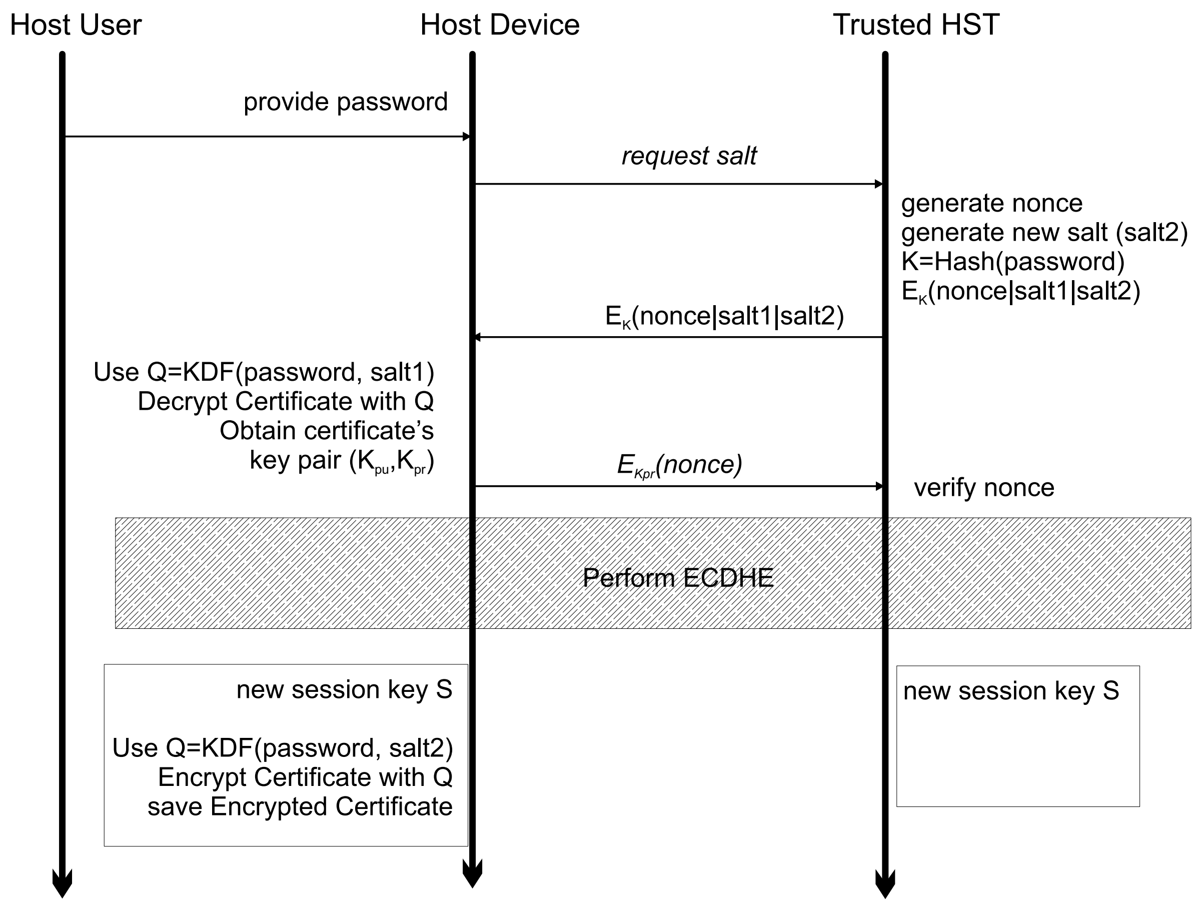

Figure 3. The proposed session key agreement protocol

security, this value cannot be reused. This is the reason behind the need for a new salt value (salt2). After receiving $E_K(nonce|salt1|salt2)$ if the Host has the correct password, he can decrypt the recovered message, extract the salt (salt1) and through a KDF recreate the key Q that is necessary to access the stored user's certificate and keys. Using these Asymmetric Encryption Keys, the host then sends a digital signature of the nonce to the HST thus verifying knowledge of both the nonce and his private key ($E_{K_{pr}}$). Then, the ECDHE key agreement scheme is executed, leading to a common AES session key S. The Certificate and Asymmetric cryptography keys are encrypted using the result of a KDF that has as inputs the password along with the new salt (salt2). This result is stored back to the Host storage area.

After establishing a secure communication between the HST and the Host, the Host (using the HST/Host software component) can transmit to the HST data that need to be forwarded with integrity and authenticity through the HST Ethernet IP communication. This can be achieved by digitally signing the data with the HST Asymmetric Cryptography keys. Finally, the HST can be extended to operate as an ADS sensor by monitoring the CAN bus (for CAN bus CI based systems) and transmit such data again through its Ethernet channel.

4.3. Host-HSM logging mechanism

When a security related incident is taking place, it can be detected by the HST/Host software component. A log entry is then generated to either be stored using the *syslog* protocol in the auth.log of the Linux OS as a syslog entry in the Host device or been generated internally in the HST/Host software component and been forwarded if confidentiality, integrity and authenticity to a remote ADS through the HST. Such security incidents can range between possible cryptanalytic attacks, loss of connection between the Host and the HST, password authentication failure and many others. The proposed logging mechanism ought to be flexible and simple to use, while containing sufficient amount of information that can be scaled up according to the application's needs. Through this scope, each log entry is a JSON message that includes information related to a specific event in the following format:

```
{
    "HostID":<integer>,
    "HostIP":<integer>,
    "HostState":<string>,
    "HSTid":<integer>,
    "timestamp":<integer>,
    "event":{
        "type":<integer>,
        "failure":<integer>,
        "severity":<integer>
    }
    "comments": <string>
}
```

The above structure is always digitally signed with the keys that are stored inside the HST (the Host/HST software component doesn't have access to them). Note that the above log format can be expanded with relative ease to include additional fields of varying type, producing a more detailed log entry that conveys a complete overview of an event with adequate information. The first five basic fields of this JSON array format are necessary for the correct identification of the specific unit that produces a log entry, as well as the exact time the log was generated. The *"HostState"* field provides characterization of the Host state relative to the HST. There exist two states on which a CI Host can be in, administration and user states. In the administrator (*admin*) state, the Host user can gain full access to the HSM services and features, including the ability to store other Host Entries to the HST flash memory, as was analyzed previously. In the *user* state, however, the Host user lacks the authorization to add new Host Entries to the HST. The core of the log entry containing the most important information is the *"event"* array. Its first field, *"type"*, is an integer number indicating the type of event that has been logged. In its current HST realization, the acceptable values of this field are the following types:

- 0: Message integrity validation event.
- 1: Password based Host to HST session initiation.
- 2: HST availability.
- 3: Security channel failure.

Directly linked to the above, the field *"failure"* is an integer number that indicates if the occurred event of the type specified in *"type"* has failed (*failure*=1) or if it has concluded normally (*failure*=0). The last field encapsulated in the *"event"* is the *"severity"* one, signifying the importance in terms of security impact in cases where event failure is detected. A value of 0 indicates low severity, while the maximum value of 3 marks the logged event with the highest severity. In the final JSON field *"comments"*, additional information related to the logged event can be provided. After generation, the log entry is sent to the ADS monitoring system, containing all the necessary information about the Host device, the logged event and its severity. Accordingly, the level of trust and confidence by the ADS is enhanced, providing simultaneously valuable details that aid the appropriate management of the inflicted node or device.

5. HST Practical Conceptualization-Realization

In order to further promote the functionality and applicability of the HST concept and highlight its importance as a flexible and scalable system that provides solutions to many different security problems CI systems along with trustworthy anomaly detection, in this section we describe a realization of the proposed solution that was implemented during the EU project "CIPSEC:Enhancing Critical Infrastructure Protection with innovative SECurity framework" and is been expanded in EU projects "CONCORDIA" and "CPSoSaware". In this realization we specify the HST CLI environment and show its practicality in promoting and accessing the HST security services. Consistency and ease of upgrading should be essential characteristics of this CLI, focusing on scalability and adaptability to a wide range of security scenarios that need to be implemented in multiple CI Systems. These scenarios can vary from simple End-to-End secure channel establishment, to appropriate secure and trusted logging and even to a PKI-like scheme that manages Host public keys. In its current realization, the HST is implemented in a Digilent Zedboard device that includes the Zynq 7000 series SoC with ARM Cortex A9 processor and an FPGA fabric on chips (used for the implementation of the HST Hardware IP cores)

5.1. Case study HST CLI for Cryptographic Application Programming

The availability of a variety of IP cores, as well as the sufficiently powerful Cortex-A9 processing unit offer the ability of implementing numerous cryptographic and security protocols available today. These operations can be accessed directly from the Host machine through an in-house built serial CLI. The serial console component accepts commands for execution that adhere to a specific format. Its general structure is as follows: *"command [options] [HostID] [data]"*. For example, in order to execute the HMAC (Hash-based Message Authentication Code) protocol, the Host sends to the HST through the serial secure communication channel the command *"hmac [key] [message]"*, where [key] is a secret shared key and [message] the input data of the algorithm. In cases when a command requires extra information related to a specific Host like an ECC public key, an extra *[HostID]* field must be added, providing to the HST the ability to extract and use the correct Host Entry located in its flash memory.

The HST's hardware board cryptographic features can be accessed through the HST/Host software component. This tool is developed for Linux OS based Host machines and is currently realized for 32 bit and 64 bit x86 Linux platforms as well as ARM Linux platforms. During operation, two different modes are available. In the HST Console mode the various CLI commands are transmitted directly to the HST through a terminal console. In the HST OS mode, on the contrary, all HST commands are given as arguments upon our Linux based HST/Host software component executable or as CLI scripts. As it is apparent, OS mode provides greater functionality and flexibility, enabling the development and easy deployment of different applications that want to take advantage of the HST's features. The overall CLI approach that is been used resembles the openssl library approach with the extension of hardware, dedicated trusted tokens use (a hardware in the loop concept i.e the HST).

5.2. Exemplary Deployment and various use cases

In a plethora of industrial and CI systems, sensitive information is being exchanged continuously by a wide range of different Host machines. This exchange is often exposed to attacks both in a physical and network level, as many hardware sensors and Host devices operate on a variety of hostile environments. This is even more prevalent in many Legacy systems that inherently lack strong security design principles. Thus, in such cases the HST functionality can be extended to offer support of certificate management for the different Host devices a CI utilizes, promoting a unique Host-HST module as a pseudo-CA (Certificate Authority).

For this purpose, the HST is modified to include a Raspberry Pi module connected to the Zynq 7000 series FPGA board (Zedboard) through a USB serial communication channel. This Pi module, connected to the IP network with either Ethernet or Wi-Fi connection emulates a standalone Host machine. In a similar manner, we assume there exist identical Host-HST pairs operating across a CI. Such a configuration can be seen in Figure 4.

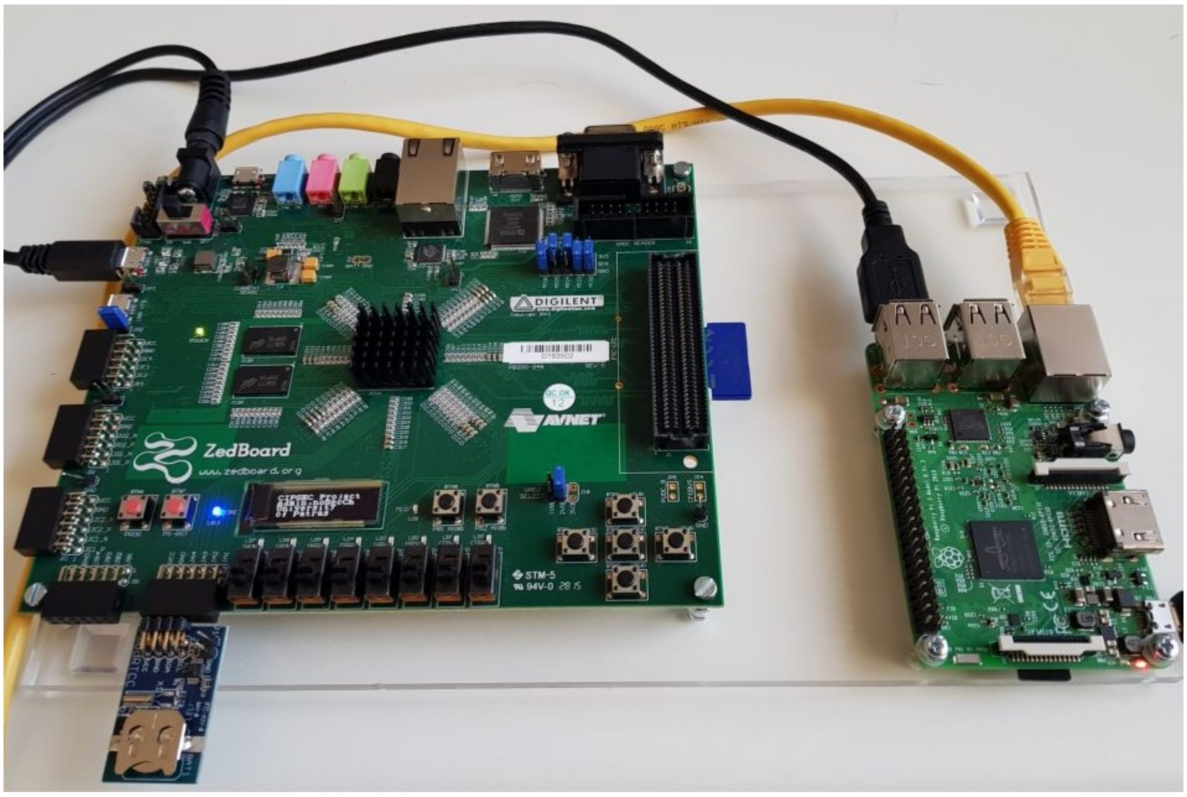

Figure 4. Raspberry Pi and Zedboard Host-HSM configuration

Typically, in this scenario, most of the Host devices operate in *user* mode, meaning they can't add another Host Entry in the HST's flash memory other than their own entry during Host initialization phase due to lack of elevated privileges. Each Host device has already generated and stored a Host ECC private and public key pair. Acting as a pseudo-CA, a Host-HST module is operated in *admin* mode. As already mentioned, a Host in *admin* state is authorized to store additional information on the flash memory module embedded in its corresponding HST. With the functionality our Crypto-Library offers, the CA can receive Certificate Signing Requests (CSR) from any *user* Host operating in the same network as the CA. The CA then checks the validity of the digitally signed CSR, and upon successful validation generates an ECC Certificate bound to the specific *user* Host. The Certificate is stored on the CA's corresponding Host Entry and sent through the network to the Host that requested it. Through this process, the CA is in possession of all the user Hosts' certificates and updates this list whenever a new Host is added to the network or a Host regenerates its ECC key pair. Any Host from this point forward can request from the CA another Host's certificate, in order to validate the authenticity of its public key and consequentially the ownership of the corresponding private one. Using this PKI-like structure, greater trust is instilled upon the different Hosts' communication than utilizing a basic digital signature scheme without the existence of a CA.

An additional use case that is quite frequent in many CI systems is the ability for a security component to provide End-to-End encryption, integrity and authentication. Extending our system's functionality, the Host-HST pair is able to capture and redirect UDP packets destined for another Host device. In this mode, however, the Host is exclusively communicating with its associated HST that is responsible for handling the network traffic related to the Host machine. The source of this traffic (UDP packets) that is forwarded to the Host's HST, can either be encrypted information from another Host - HST pair or data that need to be encrypted before being forwarded to a destination Host - HST pair (in order to achieve end to end secure communication). For a mechanism like this to operate properly, the involved HSTs (one for each end point Host) execute a key exchange protocol to generate a common shared session key. After this establishment, the raw data a particular Host wishes to send to another Host must firstly be encrypted and authenticated by the HST (using for example authenticated encryption or encryption and MAC mechanisms) and attached to a UDP packet. The UDP packet is then sent through the network accordingly to the correct destination Host - HST pair, where the ciphertext is decrypted and its authenticity - integrity validated, thus revealing the original raw data to the destination Host. Utilizing this design philosophy, two Hosts can effectively establish a secure channel of communication even over IP, taking advantage of the encryption and decryption services only the HSTs can provide. The logging mechanism is constantly active in the above mentioned activities. This means that if any failure in the overall process is detected (e.g wrong key establishment, no authentic message, non authentic Host user that tries to access the HST etc.) then the ADS monitoring system is informed through the HST dedicated Ethernet reporting channel.

6. Conclusion

In this paper, we propose an approach on how to collect information from CI device ADS sensors that can be trusted and are not tampered with. This approach was based on a hardware assisted dedicated security service provider, the HST, that supports a secure event log and monitoring mechanism. In our approach, the goal is to move securing of security related logs, needed by an ADS, from the Operating System of a CI host (that can be considered insecure) to the HST dedicated hardware module. The HST performs operations in a secure environment and has sole knowledge of cryptography keys that are used for providing confidentiality, integrity and authenticity of the logging mechanism. Thus, even if an attacker manages to compromise the CI Host system, he still doesn't have knowledge of the security keys and also doesn't have access to the log monitoring mechanism (which in our proposal is fully manifested in the HST). In the paper, we analyzed the proposed approach based on the above described concept, detailed the HST functionality as well as the functionality of the associated HST-Host security component deployed in the Host device. We also described how the log mechanism could be realized (using JSON data structures) and also provided a practical realization of the HST concept. After describing a manifestation of the HST command line interface, we also described cases study scenarios where the HST can be used to provide even additional services to the secure log monitoring and reporting mechanism. As future work we plan on realizing an ADS CAN based sensor inside the HST and collect CI industrial type of sensor data within the HST. Eventually, we are aiming at minimizing the role of the HST-Host software component and create a self-contained

HST device that can work autonomously (without a host) inside the CI infrastructure so as to also report network-based security events.

Author Contributions: The authors of the paper have contributed to the work as follows: conceptualization, Apostolos Fournaris and Charalambos Dimopoulos; methodology, Apostolos Fournaris and Charalambos Dimopoulos; software, Apostolos Fournaris and Charalambos Dimopoulos; validation, Apostolos Fournaris, Konstantinos Lampropoulos and Charalambos Dimopoulos; formal analysis, Apostolos Fournaris and Charalambos Dimopoulos; investigation, Apostolos Fournaris and Charalambos Dimopoulos; resources, Apostolos Fournaris and Odysseas Koufopavlou.; data curation, Apostolos Fournaris and Charalambos Dimopoulos; writing—original draft preparation, Apostolos Fournaris and Charalambos Dimopoulos; writing—review and editing, Konstantinos Lampropoulos; visualization, Konstantinos Lampropoulos; supervision, Apostolos Fournaris and Odysseas Koufopavlou; project administration, Odysseas Koufopavlou.; funding acquisition, Apostolos Fournaris and Odysseas Koufopavlou.

Funding: This paper's work has received funding from the European Union's Horizon 2020 research and innovation programme CPSoSaware under grant agreement No 873718 and also the European Union's Horizon 2020 research and innovation programme CONCORDIA under grant agreement No 830927).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fournaris, A.P.; Lampropoulos, K.; Koufopavlou, O. Trusted hardware sensors for anomaly detection in critical infrastructure systems. 2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), 2018, pp. 1–4.
2. Mitchell, R.; Chen, I.R. A Survey of Intrusion Detection Techniques for Cyber-physical Systems. *ACM Comput. Surv.* **2014**, *46*, 55:1–55:29. doi:10.1145/2542049.
3. Coppolino, L.; Kuntze, N.; Rieke, R. A Trusted Information Agent for Security Information and Event Management. *ICONS 2012 Seventh Int. Conf. Syst.* **2012**, pp. 6–12.
4. King, D.; Orlando, G.; Kohler, J. A case for trusted sensors: Encryptors with Deep Packet Inspection capabilities. MILCOM 2012 - 2012 IEEE Mil. Commun. Conf. IEEE, 2012, pp. 1–6. doi:10.1109/MILCOM.2012.6415703.
5. Zuech, R.; Khoshgoftaar, T.M.; Wald, R. Intrusion detection and Big Heterogeneous Data: a Survey. *J. Big Data* **2015**, *2*, 3. doi:10.1186/s40537-015-0013-4.
6. Fraile, L.P.; Fournaris, A.P.; Koufopavlou, O. Revisiting Rowhammer Attacks in Embedded Systems. 2019 14th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS), 2019, pp. 1–6. doi:10.1109/DTIS.2019.8734936.
7. Fournaris, A.P.; Pocero Fraile, L.; Koufopavlou, O. Exploiting hardware vulnerabilities to attack embedded system devices: a survey of potent microarchitectural attacks. *Electronics* **2017**, *6*, 52.
8. Fournaris, A.P.; Keramidas, G. From hardware security tokens to trusted computing and trusted systems. In *Syst. Des. Methodol. Telecommun.*; 2014; Vol. 9783319006, pp. 99–117. doi:10.1007/978-3-319-00663-5-6.
9. Group, T.C. TCG TPM specification version 1.2, 2007.
10. Challener, D.; Yoder, K.; Catherman, R.; Safford, D.; Van Doorn, L. *A practical guide to trusted computing*; IBM press, 2007.
11. Intel. Intel® Trusted Execution Technology (Intel® TXT) **2011**.
12. ARM. ARMTrustZone.
13. Hein, D.; Toegl, R. An Autonomous Attestation Token to Secure Mobile Agents in Disaster Response. The First International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MobiSec 2009). Torino, 2009.
14. Fournaris, A.; Lampropoulos, K.; Koufopavlou, O. Hardware Security for Critical Infrastructures - The CIPSEC Project Approach. Proc. IEEE Comput. Soc. Annu. Symp. VLSI, ISVLSI, 2017, Vol. 2017-July. doi:10.1109/ISVLSI.2017.69.
15. Heiser, G.; Andronick, J.; Elphinstone, K.; Klein, G.; Kuz, I.; Ryzhyk, L. The road to trustworthy systems. Proceedings of the fifth ACM workshop on Scalable trusted computing. ACM, 2010, pp. 3–10.
16. Fournaris, A.P.; Koufopavlou, O. Protecting CRT RSA against Fault and Power Side Channel Attacks. 2012 IEEE Comput. Soc. Annu. Symp. VLSI. IEEE, 2012, pp. 159–164. doi:10.1109/ISVLSI.2012.54.

17. Fournaris, A.P.; Zafeirakis, I.; Koulamas, C.; Sklavos, N.; Koufopavlou, O. Designing efficient elliptic Curve Diffie-Hellman accelerators for embedded systems. 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 2015, pp. 2025–2028. doi:10.1109/ISCAS.2015.7169074.