


## Article

# Safe and Policy Oriented Secure Android-Based Industrial Embedded Control System

Raimarius Delgado <sup>1</sup>, Jaeho Park <sup>1</sup>, Cheonho Lee <sup>2</sup> and Byoung Wook Choi <sup>1,\*</sup>

<sup>1</sup> Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, Korea; raim223@seoultech.ac.kr (R.D.); jaeho@seoultech.ac.kr (J.P.)

<sup>2</sup> JECS Company Incorporated, Seoul 01811, Korea; joel@jecs.co.kr

\* Correspondence: bwchoi@seoultech.ac.kr; Tel.: +82-02-970-6412

Received: 17 March 2020; Accepted: 14 April 2020; Published: 17 April 2020



**Abstract:** Android is gaining popularity as the operating system of embedded systems and recent demands of its application on industrial control are steadily increasing. However, its feasibility is still in question due to two major drawbacks: safety and security. In particular, ensuring the safe operation of industrial control systems requires the system to be governed by stringent temporal constraints and should satisfy real-time requirements. In this sense, we explore the real-time characteristics of Xenomai to guarantee strict temporal deadlines, and provide a viable method integrating Android processes to real-time tasks. Security is another issue that affects safety due to the increased connectivity in industrial systems provoking a higher risk of cyber and hardware attacks. Herein, we adopted a hardware copy protection chip and enforced administrative security policies in the booting process and the Android application layer. These policies ensure that the developed system is protected from physical tampering and unwanted Android applications. The articulation of the administrative policies is demonstrated through experiments. The developed embedded system is connected to an industrial EtherCAT motion device network exhibiting operability on an actual industrial application. Real-time performance was evaluated in terms of schedulability and responsiveness, which are critical in determining the safety and reliability of the control system.

**Keywords:** safety; real-time; policy oriented security; android; industrial control systems

## 1. Introduction

Industrial control systems (ICS) comprise heterogeneous hardware and software components for control, sensing, and actuation, that are interconnected with specialized industrial networks. These are typically employed in environments for industrial control and automation. Owing to the remarkable advances in technology, ICS has expanded to various application domains including energy and power systems, transportation, avionics, and robotics [1–4]. Although traditional ICS main controllers are based on high-end powerful computers, in the last decade, the rapid development of embedded systems and their consistent increase in performance has made them valid as strong alternatives. Indeed, embedded systems are able to provide better solutions, especially in practical applications, owing to their portability, low power requirements, and relatively inexpensive costs compared to their high-end counterparts. This paradigm shift enables the smaller and low-cost design of control systems while remaining competitive in terms of performance. Embedded systems as the main controllers in various cyber-physical systems for industrial automation are also considered as one of the backbones of the fourth industrial revolution [5]. ICSs are “safety-critical” systems, which means that any task failure can lead to catastrophic results that can damage properties, infrastructures, or even people. Thus, ICSs should be bounded with stringent temporal constraints and should guarantee hard real-time performance [6] to avoid such accidents or malfunction.

Due to the complex combination of numerous hardware devices and control algorithms, the underlying software should be robust to any changes. Thus, the traditional super-loop concept [7] or single task software development is not advisable. Operating systems were introduced, offering standard application programming interfaces (API) to ensure easier interaction with different devices in various application domains. They also provide a multi-tasking environment, separating functions into self-contained processes, executed depending on the governing scheduler. Operating systems are divided into two major classifications. General purpose operating systems (GPOS) are more concerned on the best-effort performance of the system with more relaxed timing. In contrast, real-time operating systems (RTOS) conserve strict timing in executing tasks, which are scheduled based on priorities. Simply put, tasks with lower priorities can run only if there are no available high-priority tasks. This behavior guarantees the determinism and predictability of executing real-time tasks, which are critical requirements in satisfying the safety of ICSs.

Android is gaining huge popularity as the leading operating system of embedded systems owing to its open source nature. Android was developed as a software stack on top of the Linux kernel. As Android is a GPOS, its ability as the main controller of an ICS is still in question. Currently, there is a huge effort from various researchers to evaluate the viability of Android and its utilization in real-time embedded and industrial applications [8–10]. According to these studies, the feasibility of Android-based industrial controllers is still an open problem due to two major drawbacks: safety (real-time characteristics) and security. Several studies have addressed the timeliness and real-time constraints of Android. Yan et al. [11] proposed RTDroid, a real-time variant of Android with most of the internal components redesigned. This provides a deterministic response of Android applications by replacing both the kernel and runtime with their respective real-time alternatives. RTDroid was tested mainly for real-time sound processing, and the measured sound latency performance is presented in their extended work [12]. However, its extendibility to industrial applications is moot as most industrial software tools and libraries are designed for the standard Linux kernel. Porting these libraries to Android requires significant amount of development costs and time. In this regard, the most common approach is running Android on a remote guest system connected to a host controller with an RTOS. Truong et al. [13] presented the remote monitoring and control of an industrial CNC machine via a wireless network on an Android platform. This is remarkably similar to the solution of Mateo et al. [14], called Hammer, an Android-based teach pendant application, to control industrial robotic arms. These approaches can cause performance bottleneck on the host system, depending on the governing communication protocol. Manufacturing cost is also an issue because these solutions require more than a single hardware platform. To this end, we aim to address the safety issues of Android by developing a single-board embedded system which can guarantee hard real-time constraints and integrate Android applications to real-time tasks.

Security is another issue that affects safety due to the increased connectivity in industrial systems, which causes a higher risk of cyber and hardware attacks. In an ICS, attacks are divided into three major categories: physical, communication, and software attacks. Physical attacks refer to any change in the hardware that includes physical contact, such as the removal of a chip package or tampering with any devices. In communication attacks, the attacker can manipulate, intercept, or spoof any message coming from a device within a network. Software attacks are mostly executed by accessing restricted resources through malware, system bugs, or any unexpected call sequences. For embedded systems based on the ARM architecture, ARM TrustZone [15] has been widely available on selected platforms that provides protection for the software that executes within the platform. It also provides a secure boot sequence that only loads authenticated kernel images. Aside from the limited availability of supported hardware, the developers of the technology do not disclose technical details, resulting in difficulties in its implementation. Its application on Android is also questionable due to it being inaccessible directly from Android processes or applications. For Android applications, security is handled in the software stack or in the ecosystem [16]. Attacks on the software stack result in the direct acquisition of root privileges, giving the attacker complete access to and control of the entire

system. In contrast, the ecosystem form of attacks refers to the attackers tricking naive users into downloading and installing bogus applications to gain control of the device. To prevent any intrusion and corruption to the software stack (specifically the kernel), Samsung Knox offers Real-time Kernel Protection (RKP) [17]. However, RKP is highly dependent on the ARM TrustZone, along with its flaws, and the system is still vulnerable to ecosystem attacks [18]. Several researchers have addressed this issue, focusing on the different forms of ecosystem attacks, namely privilege escalation [19], advertisements [20], and application repackaging [21,22]. Although most of these studies increase the security of Android, they require changes to the native code and could exhibit additional performance overheads to the end users. In this sense, we aim to enforce security schemes ensuring that the embedded system is protected from physical tampering, and also ensure that attackers could not run unauthorized Android applications.

In this paper, we develop a safe and secure Android-based embedded system applicable as a main controller of industrial control systems. To address the safety and real-time issues of Android, the real-time characteristics of Xenomai [23], a dual-kernel approach of real-time Linux, were explored and implemented on our own developed NXP i.MX6-based embedded platform, namely JECS-600ITX. Although the Linux kernel and Android sources are provided as board support packages (BSP) by NXP, compatibility with Xenomai and other patches is trivial. Thus, we provide complete development details on developing a real-time environment, considering the version compatibility of Linux, Android and Xenomai for easier reproduction. In a dual-kernel configuration, running the Linux system calls in the Xenomai domain result to mode switching. This causes the real-time tasks to be non-deterministic, resulting in a system freeze or a kernel panic. To deal with this issue, we developed a communication interface between Android applications and Xenomai, utilizing the shared memory mechanism of Android Interface Definition Language (AIDL) and a variant of our previous work in [24]. This interface ensures integration and smooth communication between Android applications and Xenomai real-time tasks. An open source EtherCAT master was implemented on top of Xenomai [25] to connect with actual ICS networks.

Dealing with the security issues of Android, we have adopted a hardware copy protection chip called Algorithm License Permmittution Unit (ALPU) by Neowine [26]. As the chip only comes with a library supporting Linux, encryption APIs were developed for both the bootloader and Android to enforce four administrative policies. To prevent hardware breaches, such as tampering with the ALPU, the following policies are implemented in the booting process: (1) the booting process is initiated only in the presence of the ALPU, and (2) the booting process is initiated only if the stored encryption key in the embedded platform matches the ALPU. With the ultimate goal of protecting the embedded system from running unauthorized applications in Android, the following policies are enforced in the Android application layer: (3) third party software, which should include the developed authentication library, and (4) only applications with the encryption key are executable. These policies ensure that the embedded platform is protected from any physical breaches such as modifications to the firmware of the embedded system and avoids any form of unwanted software intrusion in the Android software stack. Experiments were conducted to demonstrate the articlucy of the administrative policies and the JECS-600ITX is connected to a series of industrial motion devices (EtherCAT servo drives) to show operability in an actual working environment. The real-time performance of the embedded platform is evaluated in terms of schedulability and responsiveness, which are extremely critical to determine the safety and reliability of the entire ICS.

The remainder of this paper is organized as follows. Section 2 introduces the software architecture and the details of the development environment of the Android-based industrial presented in this work. This section also includes the communication interface integrating Android applications and Xenomai real-time tasks. In Section 3, an overview of the policy-oriented security scheme and the ALPU is provided. Section 4 presents the experiments and results, demonstrating the practicality of the developed embedded system in terms of the articlucy of the administrative policies, operability on an

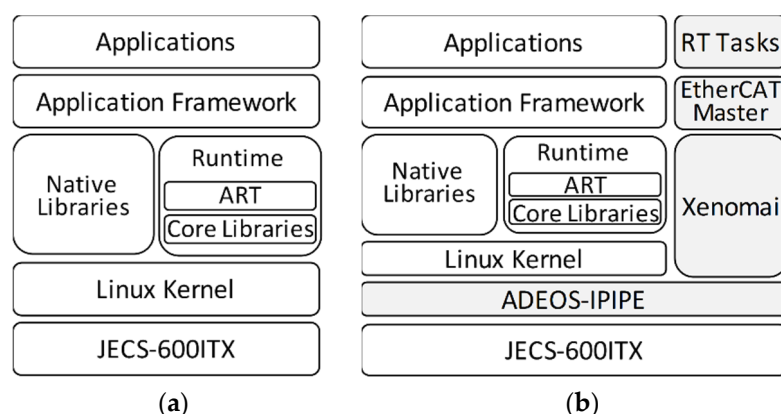
actual industrial network, and real-time performance. The last section summarizes the concluding remarks of the paper.

## 2. Android-based Industrial Controller for Safety-Critical Applications

Safety is an essential requirement for ICSs to avoid system malfunction or accidents in a working environment. In this paper, we define “safety” from a functional point of view, such that all functions must be executed correctly at the correct time or within a specific temporal deadline to be safe. This means that ICSs should adhere to hard real-time constraints to avoid system failure. To meet such requirements, the main controllers of ICSs were traditionally developed based on powerful computers with high-performance RTOSs. As the current trend of developing main controllers based on embedded systems becomes a paradigm, there is an increasing demand for Android in industrial applications. Researchers have evaluated and proposed various solutions to enable its use as a platform for safety-critical and real-time applications [11–14,27–29]. In this section, we provide the complete development details of a real-time environment employing Xenomai as the dual-kernel approach of real-time Linux and the implementation of an open-source EtherCAT master for our own developed embedded platform, namely JECS-600ITX. Due to mode switching [23,24], a communication interface integrating Android applications with Xenomai real-time tasks is also introduced.

### 2.1. Software Architecture

Figure 1 shows the simplified software architecture of the Android operating system for our own developed embedded system, namely JECS-600ITX. As depicted in Figure 1a, the standard Android software consists of the runtime and libraries, the application framework, and the application layers and the Linux kernel. For the Android with versions later than 5.0, the runtime consists of the core libraries and the Android Runtime (ART). Each Android application runs its own ART and runs multiple virtual machines optimized for a minimal memory footprint. Core libraries provide support for the Java programming language. Developing Android applications requires system components and services provided by the native libraries, which are written in C and C++. The Android framework layer exposes these native libraries to be used in applications. The standard Linux kernel was modified to handle the rest of the Android layers and does not support any real-time features, as the scheduler values “fairness” over priorities of processes [30].



**Figure 1.** Simplified model of Android: (a) Standard Android; (b) Xenomai-integrated Android.

The issues of real-time for Android attract various researchers. Maia et al., in [27], evaluated the real-time capabilities of Android and presented different architectural models including the usage of an RTOS kernel instead of the Linux kernel and the possibility of developing a real-time hypervisor or a real-time Java machine. In this regard, RTDroid was developed by Yan et al. [11]. However, its extendibility to industrial applications is an open problem considering that industrial software tools and libraries are mostly designed for compatibility with Linux. Porting these to Android requires a

significant amount of development time and manpower. To address this issue, various researches implemented a host–guest approach, where an RTOS is running on the host and Android on the guest system, respectively [13,14]. This would result to an increase in manufacturing costs due to the requirement of multiple hardware platforms. Thus, we present a unique solution integrating Android applications to real-time Xenomai tasks for a single hardware, as shown in Figure 1b. Modifications to the standard Android architecture are represented by the shaded blocks.

Xenomai is an RTOS implemented as a dual-kernel approach of real-time Linux. It provides a multi-tasking environment for real-time tasks (RT Tasks) with a priority-based preemptive scheduler and inter-task synchronization mechanisms. Adaptive domain environment for operating systems (ADEOS-IPIPE) abstracts hardware interrupts and other operations enabling the existence of both Xenomai and Linux on a single hardware. ADEOS-IPIPE ensures that Xenomai holds the highest priority and should preempt any non-real-time Linux processes. Meaning, Linux only runs if there are no pending Xenomai processes. The performance of Xenomai on an i.MX6-based embedded platform similar to JECS-600ITX is presented in our previous work [31]. As EtherCAT is becoming the standard network protocol in ICSs, an open source EtherCAT master, IgH Etherlab, was stacked on top of Xenomai to establish a connection with an actual industrial network. In the literature, numerous studies are present related to the implementation and performance of EtherCAT on different variations of real-time Linux [25,32,33]. To our knowledge, this work is the first attempt on implementing EtherCAT on an Android-based embedded system. The complete details of the real-time software environment for the JECS-600ITX are organized in Table 1. As the board was developed based on the reference platform, i.MX6 SABRE-SD, the same board support package (BSP), which includes the bootloader, Linux kernel and Android, is implemented. In case of the kernel, the device tree binaries were modified, considering the physical address of the onboard random-access memory. The BSP was cross-compiled on a host machine complying with the Android open-source project requirements. The resulting Android image was flashed to the internal embedded multimedia controller (eMMC) memory of the JECS-600ITX. To implement Xenomai, an ADEOS-IPIPE patch compatible with both the Linux kernel and Xenomai itself is required, however, the patch for Linux 4.1.15 is not available in the Xenomai repository. Thus, we have modified the closest available patch source namely, ipipe-core-4.1.18-arm9, ensuring that all patch hunks are successful. After patching the Linux kernel with ADEOS-IPIPE, kernel options related to power management, kernel debugging, memory page migration, and CPU frequency scaling were disabled to ensure that the unwanted voltage and frequency scaling that can affect the real-time performance of Xenomai will not occur [31]. Thereafter, the Xenomai-enabled kernel is flashed to the JECS-600ITX.

**Table 1.** Real-time environment for the JECS-600ITX.

Software Component	Version
Toolchain	gcc-arm-android-gnueabi-hf-5.4
Bootloader	U-Boot 2015.04
ADEOS-IPIPE	ipipe-core-4.1.18-arm-9
Xenomai	Xenomai 3.0.7
Linux Kernel	Linux 4.1.15
Android	Android 6.0.1 Marshmallow
IgH Etherlab	IgH Etherlab 1.5.2

The latest official distribution of IgH Etherlab is only compatible with the earlier versions of both Linux kernel and Xenomai (earlier than 4.0 and 3.0, respectively). One advantage of open software is that the source code is available to all developers and everyone can contribute to the improvement of the project. An unofficial patch set for the IgH Etherlab is available, solving the compatibility issues with our development environment, and is currently awaiting approval for mainline merge

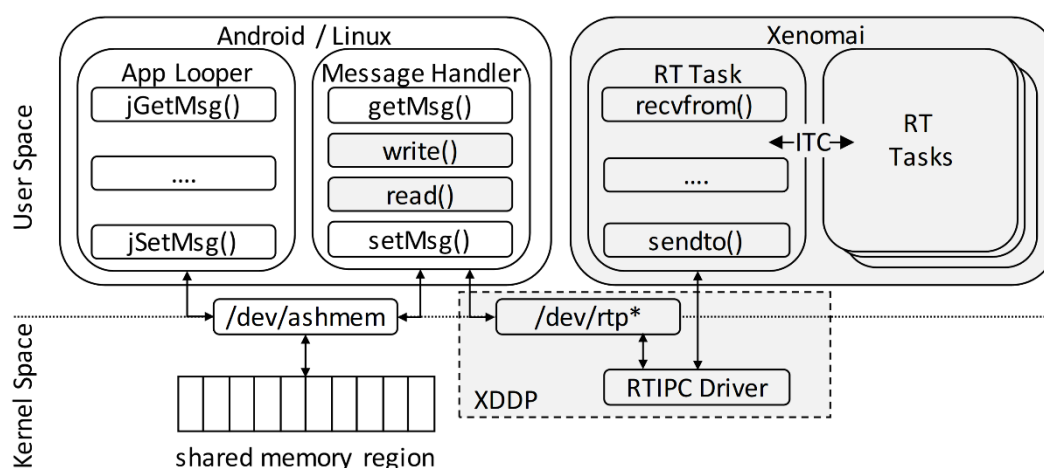


available at [34]. IgH Etherlab was compiled, enabling the generic device driver, as the real-time device drivers for i.MX6 platforms are not currently available. The user space libraries of both IgH Etherlab and Xenomai were compiled and copied to the filesystem of the Android environment. As Android does not support typical Linux operating system commands, the IgH Etherlab scripts for loading the EtherCAT master module and device drivers were modified. Instead of using `modprobe`, we employed the `insmod` and `rmmmod` commands to load and unload kernel modules, respectively. Although we have successfully leveraged a real-time environment for the JECS-600ITX, Android should be able to communicate with Xenomai tasks to be applicable in safety-critical industrial applications. The next section offers a solution, integrating Android applications with Xenomai RT tasks.

## 2.2. Integration of Android Applications and Real-Time Xenomai Tasks

In the dual-kernel configuration presented in the previous section, the entire system consists of two parts. The non-real-time part includes Android running on top of the Linux kernel and the real-time part, which is Xenomai. Accordingly, user space tasks can be divided into non-real-time Android applications, standard Linux tasks, and the Xenomai RT tasks. Each part has its own private memory address space and software abstractions, due to the hardware abstraction of the ADEOS-IPIPE. This means that direct communication between Android applications and Xenomai real-time Xenomai tasks is restricted. Moreover, using non real-time libraries or system calls within RT tasks results to an event called mode switching. This results in a chaotic scenario where RT tasks lose their hard real-time capabilities, causing the entire system to become unstable and non-deterministic. In our previous work, we implemented a cross-domain datagram protocol (XDDP) to address the communication issue between standard Linux and Xenomai [23].

Although XDDP provides communication between non real-time and real-time tasks, the same approach cannot be applied to Android due to its architecture, which does not allow direct device access to regular applications. However, Android provides an interface definition language that allows users to define a programming interface between applications to communicate through inter-process communication mechanisms. In this work, we have selected the anonymous shared memory mechanism (ashmem) as the communication channel between Android applications and standard Linux tasks. Figure 2 shows the thread model of integrating Android applications with Xenomai RT tasks, leveraging both XDDP and ashmem mechanisms. All the operations and components of the non-real-time part are represented by the white blocks, while the shaded blocks represent the real-time ones.



**Figure 2.** Thread model of integrating Android applications with real-time Xenomai tasks.

For the purpose of simplification, we assume a single connection between an Android application (App Looper) and a standard Linux task (Message Handler) via ashmem. The App Looper acts as a server, which creates a file descriptor for the shared memory and includes the message-passing

activity. Implementing the interface requires the developer to create a user-defined Android library to map the ashmem and pass file objects between processes through the Java native interface. We have created an Android library consisting of two message-passing functions to access the shared memory called `jGetMsg()` and `jSetMsg()`, which are responsible for reading and writing messages, respectively. The processing logic for these message passing functions is highly dependent on the application, which is specifically defined by the developer. In our case, we have developed the library to pass the string datatype of messages.

For the Linux side, the Message Handler is a non-real-time task that accesses the shared memory through the device file, `/dev/ashmem`, using the standard Linux `open()` function. Accordingly, we have developed `getMsg()` and `setMsg()` functions as the respective counterparts of `jGetMsg()` and `jSetMsg()` in the Linux domain. To date, communication between the App Looper and the Message Handler is successfully realized. Given that the Message Handler task should act as a bridge between Android applications and Xenomai real-time tasks, we employ the XDDP mechanism [23]. XDDP serves as the communication interface between Linux and Xenomai tasks. Herein, the Message Handler is linked to an XDDP port by accessing the real-time inter-process communication (RTIPC) driver through the `/dev/rtp*` character device (\* represents the minor number of the device file). XDDP ports are identified by this minor number and, by default, there are 32 useable XDDP ports. After opening the respective device file, the standard Linux `read()` and `write()` functions are used to access the stored message in the XDDP port. Note that the Message Handler is a cyclic task, therefore the opening of both ashmem and XDDP device files is executed before entering the infinite loop. Inside the loop, the Message Handler waits for messages from the App Looper with the `getMsg()` function and sends it directly to the Xenomai domain using `write()`. Feedback from the real-time task, acquired using `read()`, is sent back to the App Looper with `setMsg()`.

On the other hand, a Xenomai RT task accesses XDDP through socket operation. The RT task is bound to a socket with the port number, depending on the minor number of the character device accessed by the Message Handler task. This means that port 0 in the RT task is connected to the device file `/dev/rtp0`. `AF_RTIPC` and `IPCPROT_XDDP` switches are also enabled for the socket domain and protocol, respectively. To access the stored messages, `socket recvfrom()` and `sendto()` are called to acquire and set messages, respectively. In case of the `recvfrom()` function, we have enabled the `MSG_DONTWAIT` flag to enable non-blocking operation and ensure that messages from the App Looper would not affect the real-time performance of the Xenomai task. Note that this approach can be implemented in a multitasking environment, where multiple Android applications are interfaced to their respective Message Handler task in the Linux domain and RT tasks in the Xenomai domain. Xenomai also offers inter-task communication (ITC) mechanisms for synchronization and communication between RT tasks.

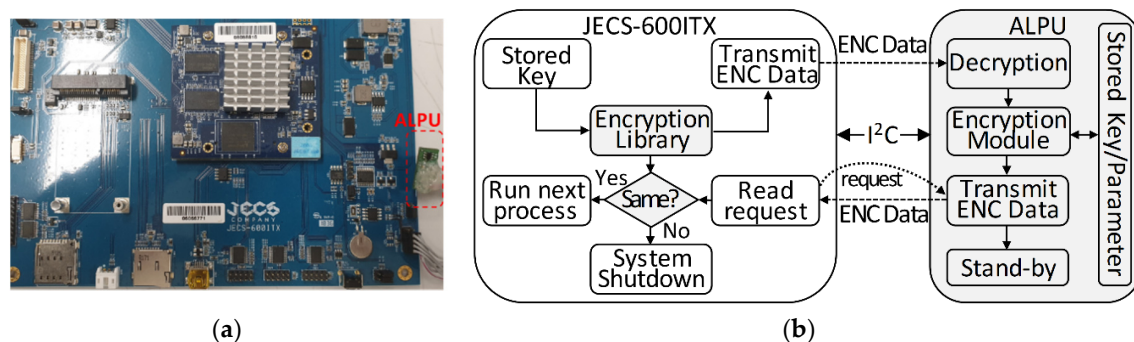
This method opens up possibilities for industrial applications that require the integration of Android with hard RT tasks. For example, safety-critical application with hard real-time constraints such as feedback control with sensors and actuators can be implemented in the Xenomai domain, while being monitored and manipulated by graphical user interface monitoring and control software in the Android domain.

### 3. Policy Oriented Security Scheme

Security is another issue that affects the safety of an ICS due to the increased connectivity in industrial systems causing a higher risk of cyber and hardware attacks. In this section, we introduce a policy-oriented security scheme employing a hardware copy protection chip called the Algorithm License Permmittuion Unit (ALPU) developed by Neowine [26]. Administrative policies were implemented in the bootloader and Android user space level to protect the developed embedded system from any form of hardware and cyber-attacks.

### 3.1. Algorithm License Permittuition Unit (ALPU)

The ALPU is a high-performance illegal copy protection chip applied in different electronic systems such as dashboard cameras, navigation systems, and industrial controllers. It is very suitable for embedded systems, considering its small size and low power requirements. The encryption module is based on a Rijndael AES-128 standard with 192-bits programmable parameters. The module consists of several blocks which are the AES-128 encryption module, the encryption parameters, feedback buffers, and the hash generator. Figure 3a shows our own developed embedded system, namely JECS-600ITX, attached with an ALPU.



**Figure 3.** JECS-600ITX with Algorithm License Permittuition Unit (ALPU): (a) actual hardware; (b) authentication process.

The ALPU offers three modes of authentication: bypass mode, feedback encryption mode, and hash generator mode. The bypass mode is only used in testing the communication between the ALPU and the main controller—in this case, the JECS-600ITX. The ALPU will perform exclusive or (XOR) operation with 0x01 on the received data and return it back to the JECS-600ITX. On the other hand, feedback encryption and hash generator are encryption methods developed by Neowine. In this paper, we focus on the hash generator mode, which is a variation of SHA-256 and is the strongest encryption algorithm available within the ALPU. Aside from the ones stored in the ALPU itself, these algorithms are distributed as a library, and used for encrypting the authentication key stored in the main controller. The authentication key is only known to the user and is unique for every ALPU chip, ensuring that the authentication process fails in cases where a different ALPU chip is attached.

Figure 3b illustrates the authentication process of the ALPU. The stored authentication key in the JECS-600ITX is encrypted depending on the selected mode of encryption. Together with the mode of encryption, the encrypted data (ENC Data) are transmitted to the ALPU through I<sup>2</sup>C protocol. Users can set the number of passes to apply the encryption algorithm, which makes the key more difficult to analyze, thereby increasing security.

The complete steps of the authentication process are as follows:

- The stored authentication key in the JECS-600ITX is encrypted using the encryption library depending on the selected mode. The encrypted data are then transmitted to the ALPU;
- Upon receipt, the ALPU decrypts the data and process them with the encryption module based on the stored key value and parameters. The results are transmitted back to the JECS-600ITX;
- Back in the JECS-600ITX, the encrypted authentication key is compared with the response from the ALPU. The authentication process is successful when both are the same;
- In contrast, the authentication fails if one of the keys is different from the other, which can invoke a system halt or shutdown depending on the procedure implemented by the developer.

Due to the encryption library being specifically made compatible only with the Linux operating system, we have ported it to be compatible with both the bootloader and Android. In the case of the bootloader, we have created a new bootloader command including the ported encryption library and



the I<sup>2</sup>C interface files to implement the authentication process as explained above. The new command is added to the list of bootloader commands and the board initiation process.

With the ultimate goal of protecting the embedded system from running unauthorized applications, we have developed an Android API including the encryption library that can be added to user-authorized applications. Note that the encryption key is compiled within the developed encryption library to prevent numerous developers from accessing it. This means that a unique library should be compiled for each ALPU. To distinguish between unauthorized and authorized applications, we have developed a security manager daemon that continuously checks whether an application contains the developed encryption API. An application without the encryption API is forcibly terminated. By implementing the encryption library ported in both bootloader and Android, we enforce administrative policies to protect the embedded system presented in the next subsection.

### 3.2. Administrative Policies

In this section, we discuss the administrative policies implemented on the JECS-600ITX attached with an ALPU. The policies were enforced with the aim of protecting the developed system from distinct types of attacks including hardware tampering, unwanted access to the device filesystem, and the execution of unauthorized applications, which are critical in operating industrial automation and control systems. These policies are implemented either before the booting process or in the Android application layer. For the booting process:

1. Booting process is initiated only in the presence of the ALPU;
2. Booting process is initiated only if the stored encryption key in the embedded platform matches with the ALPU.

The booting process of the JECS-600ITX starts when the board is connected to a power supply invoking a system startup. The first level bootloader is loaded from the CPU read-only memory to the internal memory. This initiates the CPU and the board level memory. The system bootloader stored in the eMMC is loaded in the random-access memory, which initializes the entire system. The enforced booting process policies ensure that the system checks the I<sup>2</sup>C bus for the existence of an ALPU chip. (1) If the ALPU is not present, the booting process fails invoking a system restart. If the ALPU is properly recognized, on the other hand, (2) the ALPU encryption key is compared with the one stored in the bootloader environment by implementing the authentication process through the encryption library. If the ALPU chip's credentials do not match with the one stored in the embedded system, the booting process will fail. For the Android application layer:

3. Android software and applications should include the developed encryption API;
4. Only applications with the correct encryption key are executable.

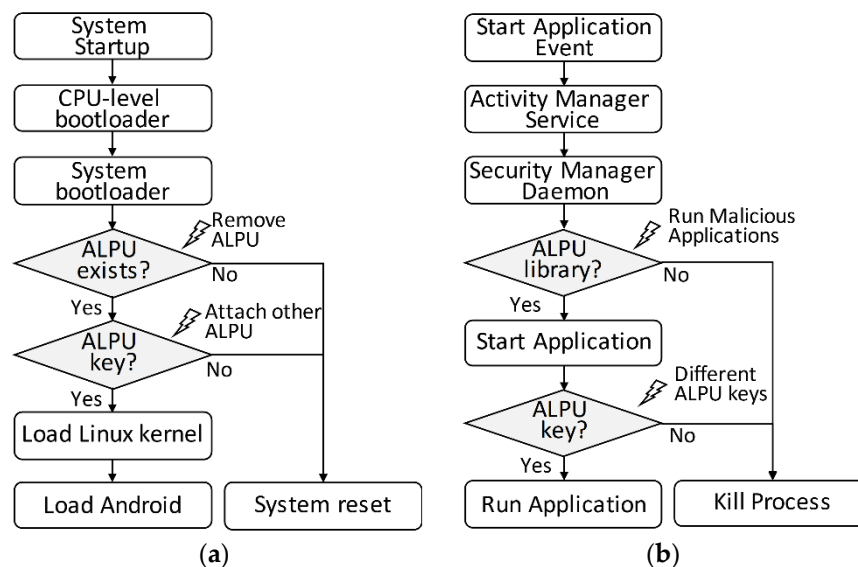
Only if the booting processes succeed will the system load the kernel and the Android operating system. The developed security manager daemon, which is responsible for checking whether applications contain the encryption API, will run as soon as Android finishes loading all essential drivers and processes. When a new Android application is opened by the user, the activity manager service performs a fork system call to the security manager daemon to run a new process. (3) If the daemon detects applications without the developed authentication library, the whole process will be killed, and the application will be terminated. Otherwise, (4) the application will start and run the authentication process with the ALPU chip. The authentication process checks whether the encryption key compiled in the Android application matches with the one stored inside the ALPU. The authentication process will fail, and the application will be terminated if the ALPU does not recognize the encryption key. This prevents execution of the application in case a different ALPU is connected to the JECS-600ITX.

In the next section, we will discuss the behavior of these policies in the presence of various threats in both the system level and the Android application layer.

### 3.3. Adversary Model

Here, we consider an adversary model where the attacker tries to physically tamper an embedded system connected to an industrial control network in order to bypass the booting process and obtain access to the Android operating system. Once the adversary has access to the operating system, all the connected devices can easily be manipulated, and confidential data such as control algorithms and machine specifications can be modified or downloaded. To achieve this goal, the adversary can either remove the ALPU chip or attach a similar one. In our developed system, we have enforced two bootloader policies, explained in the previous section, in order to prevent the adversary from booting to the Android operating system.

To restrict the adversary from tampering with any of the bootloader environment variables, we have disabled the command line interface, which results in an infinite system reset procedure. The flow diagram and the adversary model of the booting process policies are shown in Figure 4a. The standard boot procedures are depicted by the white blocks, while the shaded blocks represent the implemented booting process policies using the ALPU.



**Figure 4.** Flow diagram and adversary model of the policy oriented security scheme: (a) Booting process; (b) Android application layer.

In the figure, the adversary can try to bypass the authentication process by removing the ALPU from the embedded system. However, with the first bootloader policy enforced, the entire system will fall into an unlimited reset procedure unless it is powered off. In case the adversary can get hold of a similar ALPU chip and attach it to the embedded system, the second policy ensures that the system is protected by executing the ALPU authentication process. This means that the system will boot only if the encryption key in the ALPU is the same as the one stored inside the embedded system.

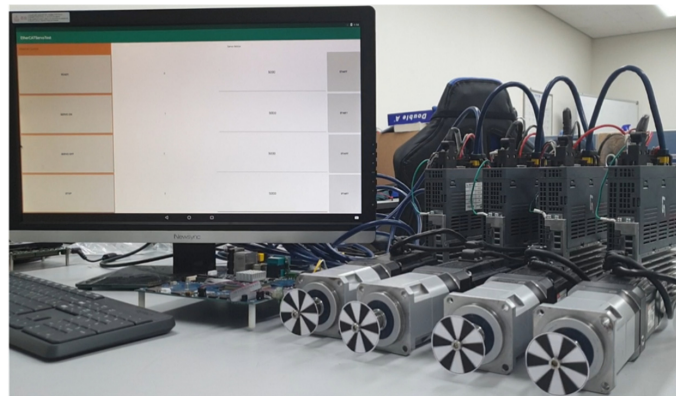
If, by any chance, the adversary is able to bypass the bootloader policies, the system is also enforced with administrative policies in the Android layer. Here, we assume an adversary trying to indirectly access the embedded system by executing unauthorized and malicious applications, as shown in Figure 4b. Through the developed security manager, the Android application is checked to ensure that it was developed with the encryption API in compliance with the third policy. If not, the security manager kills the entire process and prevents the execution of the application. If the adversary develops an application by reverse engineering the encryption API, the entire system will still be secured by the fourth security policy. The malicious application will not be executed as long as the correct encryption key is not compensated.

## 4. Experiment and Evaluation

The main purpose of the experiments is to validate the feasibility of the JECS-600ITX in industrial applications. First, to ensure that the embedded system is secured from physical and cyber infiltration, we demonstrate the articlacy of the enforced administrative policies, discussed in Section 3. Then, the embedded system is connected to a series of EtherCAT-based servo motors to prove operability on an actual industrial network. Finally, real-time performance is evaluated in terms of periodicity and the response time of the RT tasks to prove that the system is applicable in safety-critical applications.

### 4.1. Experiment Environment

To measure and evaluate the real-time environment and safety policies of the JECS-600ITX, we have tested our implementation in an actual working environment, as shown in Figure 5. The board is connected to four EtherCAT servo drives manufactured by LS Mecapion. Each servo drive was attached with AC Motors with built-in 19-bit absolute encoders. The EtherCAT slaves were configured to handle 24 Bytes of process data consisting of the status and control commands, velocity commands, and encoder feedback running in the cyclic synchronous velocity mode.



**Figure 5.** Actual working environment.

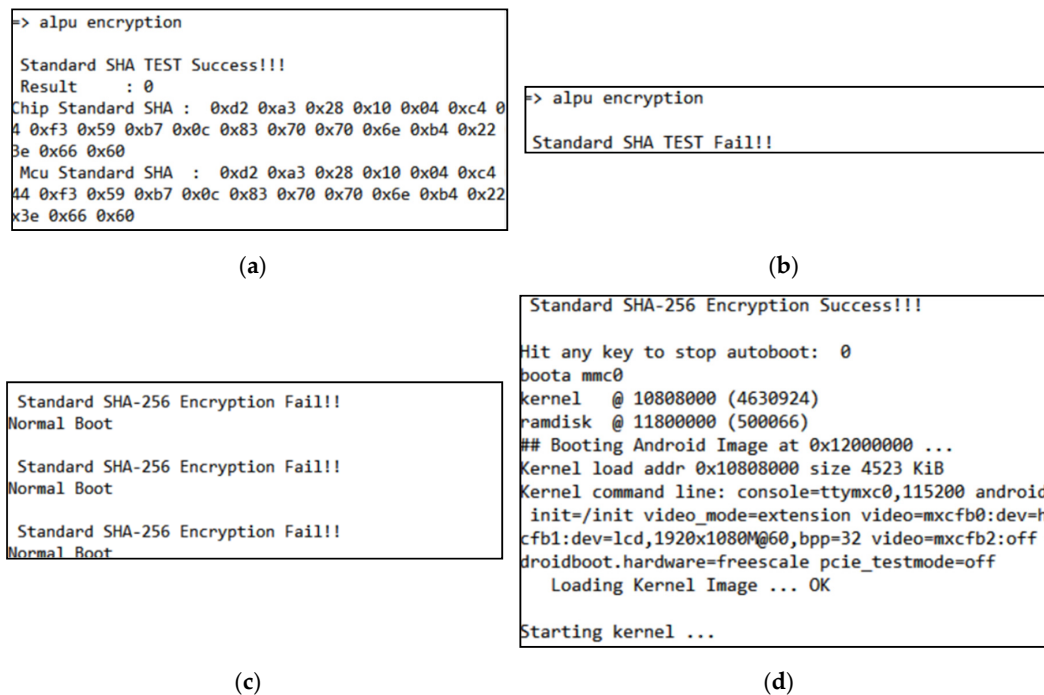
### 4.2. Articlacy of the Administrative Policies

To evaluate the articlacy of the enforced administrative policies, we conducted experiments following the scenarios given by the adversary models in Section 3.3. To demonstrate an adversary attack in the bootloader level, we performed test scenarios such as removing the ALPU and attaching another ALPU with a different encryption key. The behavior of the booting process policies is shown in Figure 6. In Figure 6a, the ALPU bootloader command line command tool was used to check the existence of the ALPU by checking the device address. As shown in the figure, the authentication process was successful, which means that the ALPU is attached to the embedded platform. When the ALPU is removed from the system, Figure 6b results in a failure, which invokes an infinite system reset without the ability to interrupt the bootloader command line in the booting process.

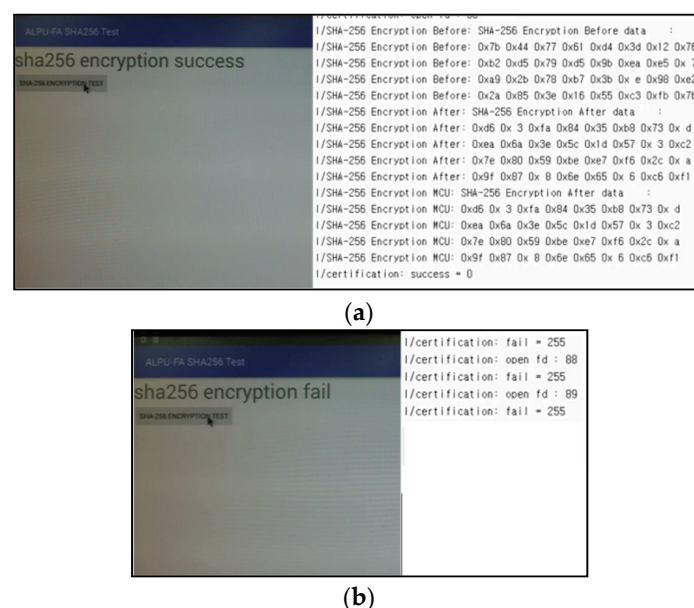
To demonstrate the second policy, the encryption key of an ALPU chip is stored as a variable in the bootloader environment. As the ALPU chip is configured to run in the hash generator mode, the key encrypted with the SHA-256 encryption algorithm should match the one stored in the respective ALPU. When another ALPU chip with a different encryption key is attached to the embedded system, the authentication process continuously fails, also leading to an infinite system reset. On the other hand, Figure 6d demonstrates a successful boot when the correct ALPU is attached to the JECS-600ITX. These results show that the bootloader security policies can protect the JECS-600ITX from any form of attack in the bootloader level.

To evaluate the articlacy of the policies in the Android application layer, we assume the worst-case scenario: that an adversary has bypassed the booting process security schemes. Accordingly,

we temporarily disabled the bootloader security schemes to boot into Android and imitate an adversary attack. Due to the fact that applications without the encryption library or the encryption key abruptly terminate, we focus on demonstrating the authentication of the ALPU key value in the Android application layer. We created a simple Android application, implementing the encryption library to verify the key value stored in the ALPU. The application encrypts the key with SHA-256 algorithm and sends it to the ALPU for verification through the I<sup>2</sup>C port. Figure 7a shows that the ALPU authentication is successful when the correct ALPU is connected to the embedded platform. However, the process fails with a different ALPU, as shown in Figure 7b.



**Figure 6.** Demonstration of the booting process policies: (a) With ALPU; (b) Without ALPU; (c) ALPU with different key value; (d) ALPU with the same key value.



**Figure 7.** Demonstration of the Android application layer policies: (a) ALPU with the same key value; (b) ALPU with different key value.

With these results, the embedded platform is protected not only at the bootloader level, but even in the Android application level. As the authentication process with ALPU chip is only performed once, when the application is executed, there is less overhead compared to the existing software security techniques in [17–22]. Implementation is also relatively easier when developing a new application because only one line of authentication function is added to the original application.

#### 4.3. Operability on an Actual Industrial Network

To validate the operability of the developed system on an actual industrial network, we have connected four industrial servo drives slaves to the JECS-600ITX via EtherCAT. A simple Android application enforced with the Android application layer security policies was created, leveraged with the shared memory procedures discussed in Section 2. The application sends the target velocity as a message to a standard Linux Message Handler task that bridges the Android application and a Xenomai RT task. The RT task runs constantly with a cycle period of 1ms and configured to the highest priority of 99 in accordance with Xenomai specifications. It is responsible for receiving the target velocity from the Android application through an XDDP port. The trapezoidal velocity profile is generated according to the target velocity received from the Android application. The task also handles the initialization of an EtherCAT master instance by utilizing the functions and services offered by IgH EtherLab.

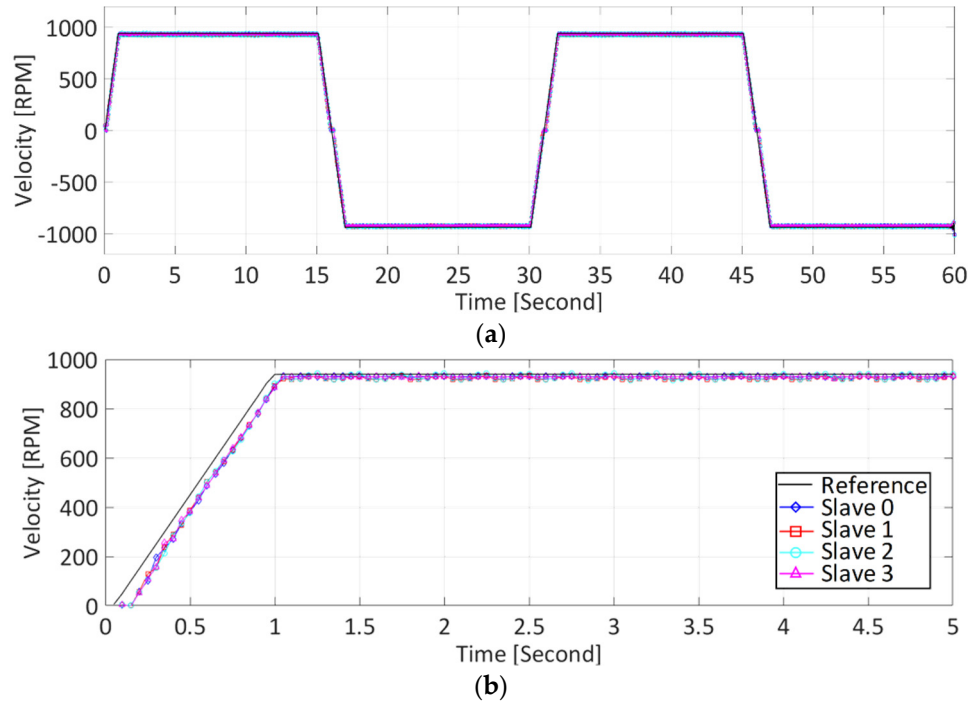
The experiment was conducted for 60 s, resulting in 60,000 samples, and the results are shown in Figure 8a. The same reference velocity was supplied to all servo drives to easily observe whether any of the EtherCAT slaves could track the given reference. We have configured the Android application to periodically send target velocities of 940 and −940 RPM every 15 s. These commands were sent to the Message Handler task, which then transmits them to the RT task via XDDP. To measure the end-to-end transaction time, we have measured the timestamps before the velocity commands are sent from Android and after they are received in the Xenomai RT task. The average transaction time for this one-way communication is 67  $\mu$ s. In the figure, the generated reference velocity profile is represented by the black solid line. All of the slaves were able to follow the reference with a minimal tracking error, as shown by the overlapping lines representing each of the slaves. In our observation, the main cause of this error is the high ratio (1:100) of the reduction gear connected to each motor. Figure 8b illustrates the first 5 s of the experiment to give a closer look of the results. Based on these results, we verified the operability of the developed Android-based industrial embedded control system when connected with an actual industrial EtherCAT network. According to Yang et al., [35] systems that meet hard real-time constraints can track a given reference accurately in comparison to their non real-time counterparts. Thus, these results verify that the system can satisfy real-time requirements, which is essential in determining the safety operation of industrial control systems. For the quantitative results of the real-time performance, we have performed real-time analysis in the next subsection.

#### 4.4. Real-time Performance

An important metric to evaluate the real-time performance of the system is the periodicity and response times of the RT task. Conducting the same experiment in the previous section, the periodicity and response times of the RT task were measured following the detailed evaluation procedures presented in our previous works [25,36]. The periodicity and response time of an RT task determines whether the task is schedulable or not. The response time is defined as the duration from the start of the task until all the required jobs are done. The jitter is defined as the difference between the configured task period, in this case 1 ms, and the actual measured value. In hard real-time applications that are safety-critical, such as industrial control, all the tasks should be schedulable and deterministic to avoid unwanted accidents and ensure safety. During the experiment, the system is kept isolated from the data acquisition and processing systems to avoid unwanted data perturbation (noises and interrupts), which could affect the validity of the results. The timing data were measured and stored in a buffer for offline processing/analysis. The samples were captured when the system was in steady-state,



discarding the EtherCAT master–slave configuration and synchronization stage. We focus all the measurements of the real-time EtherCAT control task. The results of the timing analysis are shown in Table 2 with the statistical average (avg), maximum (max), minimum (min), and standard deviation ( $\sigma$ ) values of each timing metric.



**Figure 8.** Comparison between the reference velocity and the encoder feedback: Span of (a) 60 s; (b) initial 5 s.

**Table 2.** Real-time performance of the JECS-600ITX for an EtherCAT control task.

EtherCAT Control (99, 1 ms)			
Task	Period (ms)	Response ( $\mu$ s)	Jitter ( $\mu$ s)
avg.	1.000000	91.017	0.968
max.	1.157667	1073.000	159.000
min.	0.841000	75.333	0.000
$\sigma$	0.001645	9.944	1.330

The results show that the RT task was able to run periodically with an average of 1 ms. The response time also shows an average of approximately 91  $\mu$ s, which is the overall time duration of receiving the target velocity from the Android application, generating the instantaneous velocity command, and the EtherCAT communication processes. Due to the unavailability of a real-time device driver for the JECS-600ITX, we have selected the generic EtherCAT driver that uses standard Linux system calls. This results in the maximum response time exceeding the required period (1073  $\mu$ s), which is vital in a multi-tasking environment. However, according to a presentation in ROSCON 2019 entitled Safety in Time [37], guaranteeing deadlines is typically impossible in most modern systems, and they should be given as probabilities. In this sense, the three-sigma probabilistic limit of the jitter is considered exceptional with approximately 4  $\mu$ s. Depending on the real-time application, developers can improve the response time by attaching an external network interface card with the necessary real-time driver [25,36], or by developing a real-time device driver for the on-board Ethernet controller, which is outside the scope of this paper. Considering the average response times, we can determine that the average CPU utilization of the EtherCAT control task is 0.09. According to the schedulability analysis

presented in our previous work [31] and the utilization bound test, RT tasks are schedulable if the overall CPU utilization is less than the Liu and Leyland bound, which is 1 for a singular task. Thus, the EtherCAT control task is schedulable. This also means that the developed Android-based ICS can provide a viable method of integrating Android applications with RT tasks with hard real-time requirements such as EtherCAT control.

In conclusion, all the experiment results show that the developed system adhere to hard real-time requirements, ensuring that tasks are governed by hard temporal deadlines. This behavior is essential to avoid system malfunctions and operational accidents, thus ensuring the safety of the entire system. Meanwhile, the enforced administrative security policies completely protect the ICS from various forms of cyber and physical attacks both in the bootloader and Android application level.

## 5. Conclusions

In this paper, we present an Android-based industrial embedded control system addressing the safety and security issues of Android. We defined “safety” from a functional point of view such that all functions must execute within a specific temporal deadline to be considered as “safe”. Thus, ICSs should satisfy hard real-time requirements. This paper provides a real-time environment based on integrating Android with Xenomai, the most popular dual-kernel approach of real-time Linux. Due to the hardware abstraction of the dual kernel configuration, direct communication between Xenomai and Android is restricted. Therefore, we presented a communication interface integrating Android applications with Xenomai RT task based on XDDP and the Android shared-memory mechanism, ashmem. An open source EtherCAT master was implemented on top of the real-time environment to establish a connection with an actual industrial network. To our knowledge, this is the first attempt in the literature to implement an EtherCAT master on an Android-based embedded control system. Security is another issue affecting safety considering the increased connectivity of industrial systems. We leveraged a hardware copy protection chip, namely ALPU. We created encryption libraries for the bootloader level and Android application layer to enforce the following administrative policies:

1. Booting process is initiated only in the presence of the ALPU;
2. Booting process is initiated only if the stored encryption key in the embedded platform matches with the ALPU;
3. Software and applications should include the developed authentication library;
4. Only applications with the correct encryption key are executable.

Experiments were performed on our developed embedded system, namely JECS-600ITX. Two different adversary models were formulated, demonstrating the articulacy of the enforced policies in both the system level (booting process) and the Android application layer. Operability in an industrial application was evaluated by connecting the embedded platform to an actual industrial network with EtherCAT slaves. Moreover, real-time performance was measured in terms of the periodicity and responsiveness of the entire system. Results show that the developed embedded controller displays a secure environment for safety-critical industrial applications. Our next step is to implement a real-time multi-tasking and multi-application environment which requires further evaluation of the internals of the Android architecture.

**Author Contributions:** R.D. surveyed the background of this research, developed the real-time environment and the safety scheme for the embedded control system, formulated the experiment procedures, and analyzed the results to show the benefits of the proposed controller. J.P. formulated the policies and developed the security scheme in Android and assisted in the performance evaluation. C.L. developed the industrial-grade embedded system attached with the hardware authentication chip. B.W.C. supervised and supported this study. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2019R1F1A1063547).

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

## References

1. Grau, A.; Indri, M.; Bello, L.L.; Sauter, T. Industrial Robotics in Factory Automation: From the Early Stage to the Internet of Things. In Proceedings of the IECON 2017—43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, China, 29 October–1 November 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 6159–6164.
2. Leitão, P.; Karnouskos, S.; Ribeiro, L.; Lee, J.; Strasser, T.; Colombo, A.W. Smart agents in industrial cyber-physical systems. *Proc. IEEE* **2016**, *104*, 1086–1101. [\[CrossRef\]](#)
3. Park, P.; Chang, W. Performance comparison of industrial wireless networks for wireless avionics intra-communications. *IEEE Commun. Lett.* **2017**, *21*, 116–119. [\[CrossRef\]](#)
4. Williamson, S.S.; Rathore, A.K.; Musavi, F. Industrial electronics for electric transportation: Current state-of-the-art and future challenges. *IEEE Trans. Ind. Electron.* **2015**, *62*, 3021–3032. [\[CrossRef\]](#)
5. Leitão, P.; Colombo, A.W.; Karnouskos, S. Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Comput. Ind.* **2016**, *81*, 11–25. [\[CrossRef\]](#)
6. Colnari, M.; Verber, D.; Halang, W.A. *Distributed Embedded Control Systems: Improving Dependability with Coherent Design*; Springer-Verlag London Limited: London, UK, 2008; p. 250.
7. Fischmeister, S.; Lam, P. Time-aware instrumentation of embedded software. *IEEE Trans. Ind. Inform.* **2010**, *6*, 652–663. [\[CrossRef\]](#)
8. Lam, W.; Wu, Z.; Li, D.; Wang, W.; Zheng, H.; Luo, H.; Yan, P.; Deng, Y.; Xie, T. Record and Replay for Android: Are We There Yet in Industrial Cases? In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 854–859.
9. Kang, H.; Kim, D.; Kang, J.; Kim, K. Real-time motion control on android platform. *J. Supercomput.* **2016**, *72*, 196–213. [\[CrossRef\]](#)
10. Ruan, H.; Fu, X.; Liu, X.; Du, X.; Luo, B. Analyzing Android Application in Real-Time at Kernel Level. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–9.
11. Yan, Y.; Cosgrove, S.; Anand, V.; Kulkarni, A.; Konduri, S.H.; Ko, S.Y.; Ziarek, L. Rtdroid: A design for real-time android. *IEEE Trans. Mobile Comput.* **2016**, *15*, 2564–2584. [\[CrossRef\]](#)
12. Yan, Y.; Gokul, G.; Dantu, K.; Ko, S.Y.; Ziarek, L.; Vitek, J. Can android run on time? Extending and measuring the android platform's timeliness. *ACM Trans. Embed. Comput. Syst.* **2019**, *17*, 1–26. [\[CrossRef\]](#)
13. Truong, N.; Vu, D. Remote Monitoring and Control of Industrial Process via Wireless Network and Android Platform. In Proceedings of the 2012 International Conference on Control, Automation and Information Sciences (ICCAIS), Ho Chi Min City, Vietnam, 26–29 November 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 340–343.
14. Mateo, C.; Brunete, A.; Gambao, E.; Hernando, M. Hammer: An Android Based Application for End-User Industrial Robot Programming. In Proceedings of the 2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA), Senigallia, Italy, 10–12 September 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 1–6.
15. Pinto, S.; Santos, N. Demystifying Arm Trustzone: A Comprehensive Survey. *ACM Comput. Surv.* **2019**, *51*, 1–36. [\[CrossRef\]](#)
16. Xu, A.M.; Song, C.; Ji, Y.; Shih, M.W.; Lu, K.; Zheng, C.; Duan, R.; Jang, Y.; Lee, B.; Quian, C.; et al. Toward engineering a secure android ecosystem: A survey of existing techniques. *ACM Comput. Surv.* **2016**, *49*, 1–47. [\[CrossRef\]](#)
17. Azab, A.M.; Ning, P.; Shah, J.; Chen, Q.; Bhutkar, R.; Ganesh, G.; Ma, J.; Shen, W. Hypervision Across Worlds: Real-Time Kernel Protection from the ARM Trustzone Secure World. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 90–102.
18. Kanonov, U.; Wool, A. Secure Containers in Android: The Samsung Knox Case Study. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 3–12.
19. Xu, Y.; Wang, G.; Ren, J.; Zhang, Y. An adaptive and configurable protection framework against android privilege escalation threats. *Future Gen. Comput. Syst.* **2019**, *92*, 210–224. [\[CrossRef\]](#)

20. Fang, Z.; Han, W.; Li, Y. Permission based android security: Issues and countermeasures. *Comput. Secur.* **2014**, *43*, 205–218. [CrossRef]
21. Tian, K.; Yao, D.D.; Ryder, B.G.; Tan, G.; Peng, G. Detection of repackaged Android malware with code-heterogeneity features. *IEEE Trans. Depend. Secur. Comput.* **2020**, *17*, 64–77. [CrossRef]
22. Gurulian, I.; Markantonakis, K.; Cavallaro, L.; Mayes, K. You can't touch this: Consumer-centric android application repackaging detection. *Future Gen. Comput. Syst.* **2016**, *65*, 1–9. [CrossRef]
23. Delgado, R.; You, B.J.; Choi, B.W. Real-time control architecture based on xenomai using ros packages for a service robot. *J. Syst. Softw.* **2019**, *151*, 8–19. [CrossRef]
24. Delgado, R.; You, B.; Han, M.; Choi, B.W. Integration of ros and rt tasks using message pipe mechanism on xenomai for telepresence robot. *Electron. Lett.* **2019**, *55*, 127–128. [CrossRef]
25. Delgado, R.; Choi, B.W. Network-oriented real-time embedded system considering synchronous joint space motion for an omnidirectional mobile robot. *Electronics* **2019**, *8*, 317. [CrossRef]
26. Neowine. Alpu Copy Protection Solution. Available online: <http://neowine.com/web/kor/goods/goods.view.php?gidx=154> (accessed on 15 April 2020).
27. Maia, C.; Nogueira, L.M.; Pinho, L.M. Evaluating Android OS for Embedded Real-Time Systems. In Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-time Applications, Brussels, Belgium, 6–9 December 2010; CISTER: Porto, Portugal, 2010; pp. 63–70.
28. Perneel, L.; Fayyad-Kazan, H.; Timmerman, M. Can Android Be Used for Real-Time Purposes? In Proceedings of the 2012 International Conference on Computer Systems and Industrial Informatics, Sharja, UAE, 18–20 December 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–6.
29. Zeng, X.; Li, D.; Zheng, W.; Xia, F.; Deng, Y.; Lam, W.; Yang, W.; Xie, T. Automated Test Input Generation for Android: Are We Really There Yet in an Industrial Case? In Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seattle, WA, USA, 13–18 November 2016; Association for Computer Machinery: New York, NY, USA, 2016; pp. 987–992.
30. Abbott, D. *Linux for Embedded and Real-Time Applications*; Butterworth-Heinemann: Oxford, UK, 2003.
31. Delgado, R.; Park, J.; Choi, B.W. Open embedded real-time controllers for industrial distributed control systems. *Electronics* **2019**, *8*, 223. [CrossRef]
32. Sung, M.; Kim, I.; Kim, T. Toward a holistic delay analysis of ethercat synchronized control processes. *J. Comput. Commun. Control* **2013**, *8*, 14. [CrossRef]
33. Cereia, M.; Bertolotti, I.C.; Scanzio, S. Performance of a real-time ethercat master under linux. *IEEE Trans. Ind. Inform.* **2011**, *7*, 679–687. [CrossRef]
34. Lambert, G. Igh Etherlab Repository. Available online: <https://github.com/ribalda/ethercat> (accessed on 14 April 2020).
35. Yang, G.J.; Choi, B.W. Implementation of joint space trajectory planning for mobile robots with considering velocity constraints on xenomai. *Int. J. Control Autom.* **2014**, *7*, 189–200. [CrossRef]
36. Delgado, R.; Choi, B.W. On the In-Controller Performance of an Open Source Ethercat Master Using Open Platforms. In Proceedings of the 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, Korea, 28 June–1 July 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 744–748.
37. Biggs, G. Safety in Time: Real-Time and Safety-Critical Software Development. Available online: <https://www.apex.ai/roscon2019> (accessed on 17 April 2020).

