University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

PHD THESIS

SUMMARY

Trustworthy Cyber-Infrastructure

Scientific Adviser:
Prof. Dr. Ing. Răzvan-Victor Rughiniș

Author:
Ing. Florin-Alexandru Stancu

Bucharest, 2022

Contents

1 Introduction **1**
- 1.1 Thesis Objectives . . . 2
- 1.2 Thesis Contributions . . . 2
- 1.3 Thesis Structure . . . 3

2 Trusted Computing Technologies **5**
- 2.1 Trusted Computing Group and the TPM . . . 5
- 2.2 Trusted Execution Environments . . . 5
- 2.3 Challenges in Trusted Technologies . . . 7

3 Trustworthy Cloud Services **9**
- 3.1 Current State of Trusted Cloud Solutions . . . 9
- 3.2 SecCollab - Improving Confidentiality for Existing Cloud-based Collaborative Editors . . . 10

4 Trusted Application Development **11**
- 4.1 Development Model . . . 11
- 4.2 Trusted Development Frameworks . . . 11
- 4.3 HiddenApp - Securing Linux Applications Using ARM TrustZone . . . 12

5 Embedded Devices in Trusted Infrastructures **14**
- 5.1 Evaluation of cryptographic primitives on modern microcontroller platforms 14
- 5.2 Energy Considerations Regarding Transport Layer Security in Wireless IoT Devices . . . 15

6 Trusted I/O Path **18**
- 6.1 Problem Overview . . . 18
- 6.2 Systematization of Trusted I/O Solutions for TEEs . . . 18
- 6.3 TIO - Secure Input/Output for Intel SGX . . . 20

7 Trusted Industrial Infrastructure **22**

8 Conclusions **24**
- 8.1 Thesis Summary . . . 24
- 8.2 Contributions . . . 25
- 8.3 Future Work . . . 26
- 8.4 List of publications . . . 26

Chapter 1

Introduction

Modern computers are powerful and have more than enough memory to sustain and efficiently multitask several applications. Many programs are built from millions of lines of code (e.g., graphical environments, web browsers) which, due to this complexity, makes it difficult (even unfeasible) to test and validate against security vulnerabilities. Moreover, the Operating System (OS) is usually charged with memory protection between the userspace applications and providing mechanisms for hardware / network access, interprocess communication etc. But a modern OS is also a large piece of software (e.g., Linux now has reached over 27 million lines of code [1]) which may contain privilege escalation vulnerabilities. The Trusted Computing Base (TCB) term was coined to describe the components required to be secure (e.g., hardware, firmware, hypervisor, OS kernel) for the entire system to be trustworthy. Ideally, the TCB must be kept as small as possible.

To address this problem, hardware-based trusted execution technologies started to emerge [2], allowing programs to run in secure execution contexts, isolated even from privileged system software, thus drastically reducing the TCB of the application. Popular ones include ARM Security Extensions (TrustZone [3]), Intel Software Guard Extensions (SGX [4]) and AMD Secure Encrypted Virtualization (SEV [5]). Among the possible use cases of trusted execution, we highlight: securing cloud computing services (ensuring the privacy of customer's data in case of third party hosting), embedded systems (hardening mobile / IoT / industrial devices from cyberattacks or even physical attacks), Digital Rights Management (secure software / media licensing and distribution), securing critical personal / corporate computer applications (e.g., user authentication, trusted financial transactions, protecting the confidentiality of customer databases for GDPR compliance).

Note that some of the available trusted technologies were designed for just a subset of these applications. For example, ARM TrustZone is usable only in embedded systems where such CPUs are employed. Intel's SGX technology is available on both servers and personal workstations, but mainly lacks the means to ensure a trusted I/O path with peripherals, leaving out many applications requiring secure user interaction. In contrast, AMD's SEV was specifically designed for cloud services, securing the virtual machines' memory against an untrusted hypervisor.

1.1 Thesis Objectives

Trusted execution technologies promise a refresh of the memory isolation paradigm by protecting sensitive applications against privileged actors such as untrusted OS or hypervisor. CPU manufacturers (e.g., ARM, Intel and AMD) already started to incorporate hardware-assisted isolation features into their commercial offerings, marking a major milestone to the cyber-security landscape.

Still, there are several research challenges left for Trusted Execution Environments (TEE) to be able increase their adoption rate. Thus, we define the following objectives for our thesis:

1. Study the available trusted platforms and assess their architecture, security model, practical applications and limitations;
2. Investigate solutions for ensuring the confidentiality of sensitive applications in the cloud against untrusted providers;
3. Describe the requirements of TEE application development and design methods for facilitating it;
4. Test whether modern, off-the-shelf microcontrollers are appropriate for developing trusted embedded devices (e.g., their cryptographic performance, memory, power consumption);
5. Review and develop methods for ensuring secure I/O access (e.g., keyboard, display, printers) from trusted execution environments;
6. Explore the use of trusted technologies for hardening industrial control systems' security.

1.2 Thesis Contributions

Our thesis covers the components of a trustworthy infrastructure, employing trusted execution technologies (e.g., ARM TrustZone, Intel SGX) to protect the applications' sensitive data against highly privileged attackers (e.g., compromised Operating Systems), highlighting their current limitations (applicability, portability, trusted I/O) and contributing solutions to these problems.

Summarized, our main contributions are:

- A background review of the various historic and state-of-the-art trusted technologies, including the latest commercially available solutions from ARM, Intel and AMD.
- Analysis of currently available trusted cloud technologies and our solution for ensuring the confidentiality and integrity of collaborative document editing applications (e.g., Google Docs) using transparent client-side encryption via a browser extension.

- Insight into design challenges for developing TEE code and an overview of the various SDKs and tools available for popular platforms (ARM TrustZone and Intel SGX). We develop *HiddenApp*, a system call proxying technique facilitating the migration of Linux-based applications to a TrustZone secure enclave.
- We evaluate the usability of embedded devices with resource-constrained microcontrollers in a trustworthy infrastructure, benchmarking the processing speed and energy usage with modern cryptographic libraries.
- An extensive review of the Trusted I/O Path problem and comparison of the available state-of-the-art solutions: supported peripherals, targeted TEEs and TCB components.
- The design of a portable, embedded device (*TIO*) used to establish a secure communication channel between trusted applications and USB-based peripheral devices. We demonstrate its capabilities for protecting keyboard input (e.g., passwords) and securely printing documents from an Intel SGX enclave.
- An architecture for improving the security of critical industrial control systems with trusted execution and a custom embedded firewall device used to authenticate commands sent over legacy industrial networks (using the Modbus protocol).

1.3 Thesis Structure

The thesis is structured as follows:

In Chapter 2, we introduce the trusted computing concepts and describe the major trusted execution technologies: Intel TxT, ARM TrustZone, Intel SGX, AMD SEV, plus other related research works.

Chapter 3 discusses the case of the untrusted cloud: lack of privacy guarantees when hosting applications on third party servers. As we analyze various approaches for ensuring the trustworthiness of cloud services, we argue that trusted execution technologies represent a viable solution for ensuring the confidentiality of applications processing sensitive data on untrusted servers, with experimental / upcoming support from all major cloud providers (Amazon, Microsoft, Google, IBM). We make our contribution, *SecCollab*, a method for securing the users' documents in cloud-based collaborative web applications by using client-side encryption.

Chapter 4 presents the new software development challenges introduced by Trusted Execution Environments: the need for architectural modifications, the security of interaction with an untrusted Operating System, the integration of trusted services (attestation, sealing, cryptographic implementations) and portability to other platforms. We present state-of-the-art frameworks to facilitate trusted application development or even run unmodified binaries inside secure enclaves with the help of OS abstraction libraries. Finally, we describe *HiddenApp*, our approach for easily running Linux applications inside the ARM

TrustZone's secure context by automatically proxying system calls to the rich OS for execution.

In Chapter 5, we explore the use of embedded devices as trusted platforms for specific applications (IoT, industrial and even PC security). We take several low cost, off-the-shelf microcontrollers and evaluate the performance, memory and energy consumption of various cryptographic libraries implementing modern algorithms (both symmetric and asymmetric) and protocols (TLS). We will later make use of this work to design multiple hardware prototypes that will aid in securing computer peripherals and industrial communication protocols with trusted technologies.

With Chapter 6, we introduce the Trusted I/O Path problem: due to the untrusted OS usually having privileged access over most hardware, it is now required to secure the communication between TEEs and the system's peripherals, especially when inputting or outputting sensitive data. We systematize the available state-of-the-art works in 6.2, where we analyze and compare several trusted I/O solutions targeting multiple hardware device classes and trusted platforms. In 6.3, we design *TIO* – a portable device for ensuring a trusted I/O channel between Intel SGX TEEs and USB devices. We use TIO to protect keyboard input and secure printing from enclaves.

In Chapter 7, we describe an architecture for securing industrial control systems using trusted execution, proposing a low-cost custom embedded device to act as gateway / firewall. We use it to enhance the legacy Modbus protocol with cryptographic authentication features in order to prevent attackers from sending malicious commands which can have dangerous consequences.

We conclude in Chapter 8 with the thesis summary, conclusions, future work and the list of original contributions.

Chapter 2

Trusted Computing Technologies

This chapter describes the most popular trusted computing technologies and their evolution, beginning with the Trusted Platform Module (TPM) and Intel's Trusted Execution Technology (TxT), through modern trusted execution features embedded into modern CPUs (ARM Trustzone, Intel SGX, AMD Secure Encrypted Virtualization).

Finally, we present some of the ongoing research challenges we explore throughout this thesis.

2.1 Trusted Computing Group and the TPM

The "Trusted Computing" term was promoted by the Trusted Computing Platform Alliance in 1999, later renamed to Trusted Computing Group, with the aim of making computers behave in a consistent and secure way, e.g., the ability to verify that no unintended modifications were made / malware deployed to a system before unlocking specific secrets (e.g., encryption keys, confidential data). The TCG publishes specifications for a secure computing architecture, their most popular solution being the Trusted Platform Module (TPM).

The TPM is an embeddable cryptoprocessor, often integrated into modern computing devices (e.g., PCs, laptops, servers) to improve their security. It provides a cryptographic engine, a series of Platform Configuration Registers (PCR) and small tamper-proof, non-volatile storage memory (NVRAM). These low-level features were designed to be combined together, resulting in a series of high-level security functions: integrity measurement of code and data (before execution), sealed storage (locking / encrypting data to a specific platform state) and remote attestation (proving the platform's integrity to a third party before provisioning secrets).

2.2 Trusted Execution Environments

A Trusted Execution Environment (TEE) can be described as an isolated memory region and CPU execution context where programs can run with platform-assured confidentiality and integrity protections from the normal (rich) environment, especially higher-privileged components (OS kernel, hypervisor, firmware or even physical tampering), often regarded as untrusted.

There are multiple approaches for implementing a TEE: software-only (e.g., Virtual Ghost [6], SofTEE [7]) and hardware-assisted (e.g., secure virtualization, architectural CPU modifi-

cations introducing isolated execution contexts). We focus on the latter and summarize the current commercially available trusted technologies.

2.2.1 Intel Trusted Execution Technology (TxT)

Intel's Trusted Execution Technology (TxT [8]), one of the first commercially available trusted technologies, enables a trusted hypervisor to launch late in the boot process (after the normal OS has started, excluding its entire boot chain from the platform's TCB). Intel TxT also uses an auxiliary Trusted Platform Module chip for providing integrity measurement, data sealing and remote attestation features.

This late-launch feature is said to make up a Dynamic Root of Trust for Measurement (DRTM) platform due to its integrity not being affected by any previously executed code, in contrast to the static one from the initial cold boot. Once the virtual machine has been started and measured, its resources will be isolated by the CPU's hardware virtualization protection features. The normal (untrusted) OS may continue to run side-by-side with it, outside of the system's Trusted Computing Base.

2.2.2 ARM TrustZone

ARM TrustZone [3] is an architectural extension for ARM microprocessors providing hardware-enforced separation between two domains: the Secure World, where a Trusted Execution Environment can be implemented, and the Normal World, where the rich software stack resides (i.e., the untrusted operating system and user applications). This was realized by adding a new security context, complementary to the traditional privilege levels (kernelspace and userspace). The TrustZone architecture also enforces memory isolation using an enhanced Memory Management Unit (MMU), plus a modified Interrupt Controller to provide priority interrupt handling to the Secure World. Additionally, an ARM System on Chip (SoC) may include TrustZone-aware peripherals (e.g., extra static/dynamic RAM or flash memory), which may use the execution flags to make hardware-level authorization decisions for their bus transactions.

Switching between the two Worlds can only be done by employing a special, extra-privileged exception level called Monitor Mode, where a firmware-installed handler is responsible for validating and securely saving / restoring the CPU contexts accordingly. This process happens automatically when a hardware interrupt is received and must be handled by a different security domain, or manually, when software uses the new Secure Monitor Call (SMC) instruction to access trusted services (similar to the Supervisor Call used to shift from user to kernel mode).

2.2.3 Intel Software Guard Extensions (SGX)

With **Software Guard Extensions** [4], Intel developed a new Trusted Execution Environment for their general purpose x86 CPU family. Using SGX, userspace applications may

run in special execution contexts called *Enclaves*, with hardware-level protections (including RAM encryption) against a privileged Operating System reading their memory or altering the execution flow, resulting in a minimal TCB consisting only of the CPU hardware, its microcode / embedded firmware and the enclave software.

From the user's perspective, an SGX application installs and launches the same as any other program of the Operating System. For a developer, the application must be split in two major components: the enclave code (which will be executed inside the TEE), and the untrusted program (used initially to load the enclave and provide untrusted OS services, e.g., networking, file system, peripheral access). The untrusted applications and their enclaves may switch back and forth using Enclave Calls and Outside Calls. When the enclave is loaded, its code, along with the initial data and metadata (e.g., version number, developer's public key) will be automatically hashed by the hardware inside several special measurement registers that uniquely describe the enclave running inside the trusted environment. These measurements can be used for local / remote attestation and data sealing purposes.

2.2.4 AMD Secure Encrypted Virtualization

Starting with their new EPYC (Zen-based) server CPUs, **Secure Encrypted Virtualization** (SEV [5]) is AMD's latest incursion into hardware security technologies. AMD SEV works by transparently encrypting the guest VM's RAM memory to make it inaccessible to hypervisors and even physical attackers, effectively turning virtualized instances into secure enclaves.

It uses an on-die Secure Memory Encryption (SME) controller to intercept all DRAM transactions and seamlessly encrypt / decrypt them. This can also be done in a more granular fashion by using a new *enCrypted* bit inside the page structure for enabling / disabling this process. Key generation and management happens inside the CPU's add-on Secure Processor (AMD-SP), an ARM-based microcontroller core used for trusted platform functionality. This crypto-processor also provides a firmware-based TPM and an API usable by the hypervisors for generating encryption keys to their VMs, although being unable to actually see them.

During launch, the initial image of a VM is loaded by the hypervisor as unencrypted pages. The Secure Processor also takes a cryptographic measurement of its contents and metadata, similar to a TPM's PCRs. Local VMs and remote third parties can then request an attestation report to authenticate and provision the initial secrets to the trusted VM.

2.3 Challenges in Trusted Technologies

We highlight the software / hardware tradeoff in most trusted execution technologies: hardware-based isolation approaches, although offering increased performance and resilience through better memory isolation, suffers from decreased flexibility, especially in the case of security vulnerabilities: if (or when) discovered, architectural-level bugs tend

to be hard to patch, often by using code & compiler workarounds (e.g., to prevent side channels) may lead to losses in performance. The design of a Trusted Execution Environment with a proper mix of security-validated hardware primitives and a good software (firmware) flexibility (but low TCB size) remains an ongoing challenge.

TEEs have a large applicability domain: mobile, cloud, workstations and even industrial computing. Certain hardware-based technologies are only usable in areas where such CPUs are available (e.g., AMD SEV for servers, ARM TrustZone for embedded). In our thesis, we briefly address the use of these technologies in the untrusted cloud problem in Chapter 3 (*Trustworthy Cloud Services*) and the industrial security in Chapter 7 (*Trusted Industrial Architecture*).

Next, we note that there is an added development overhead when writing TEE applications: since the Operating System is not to be trusted, any services usually provided by it (e.g., filesystem, networking, peripheral I/O, inter-process communication) either need to be secured separately, usually by employing cryptography (e.g., API results validation, using encrypted transport protocols) or, in some cases, re-implement some of them altogether (e.g., memory management, trusted clocks). This has been the target of extensive research, resulting in several approaches like OS abstraction libraries or system call validation frameworks for all commercial TEE platforms; we will describe them in Chapter 4 (*Trusted Application Development*).

Finally, one of the most outstanding issues for trusted technologies is difficulty of ensuring trusted I/O paths with the peripheral devices of a system. This is especially important for supporting interactive applications (e.g., for protecting keyboard input, display output). Some TEE technologies were designed to properly support this (i.e., ARM TrustZone for mobile devices), though this remains a challenging for general-purpose (i.e., x86) platforms. This is also requirement for industrial systems, where the cyber-physical interfaces are ubiquitous. We discuss these aspects in Chapter 6 (*Trusted I/O Problem*).

Chapter 3

Trustworthy Cloud Services

Over the last decade, the ubiquity of Internet access and the evolution of Web technologies has led to the emergence of cloud-based services and applications. However, the user's data stored there is readily accessible by the cloud providers, with important security and privacy implications. This chapter addresses the problem of the untrusted cloud and builds towards a trusted architecture for online applications.

3.1 Current State of Trusted Cloud Solutions

3.1.1 Client-side encryption

The simplest solution for ensuring data protection in a cloud environment would be to do **client-side encryption**, where the server will be unaware of the real contents of a user's data. Certain applications even allow this to be done in a transparent manner, e.g.: a third party cloud file storage / synchronization service like Dropbox may be used with a virtual filesystem that provides an encrypted view to the cloud server.

Unfortunately, this method only works when the server doesn't require to run any computation on the data for the application's features to work. Fully Homomorphic Encryption (FHE) is a promising new scheme for solving this, but, unfortunately, is not always well suited for all scenarios with its less than ideal flexibility and performance characteristics [9].

3.1.2 Trusted Cloud Platforms

Besides client-side encryption, the **Trusted Computing Technologies** have been long desired by the industry to provide affordable solution for ensuring the confidentiality and integrity of clients. Trusted Execution Environments such as Intel SGX and AMD SEV can offer increased isolation against users with privileged access (e.g., server sysadmins) by excluding the entire virtualization stack (hypervisor) and, optionally, the Operating System from their TCB. Popular cloud vendors (Google, Microsoft, IBM, Red Hat) have started to adopt them, announcing upcoming confidential cloud features to their VMs [10].

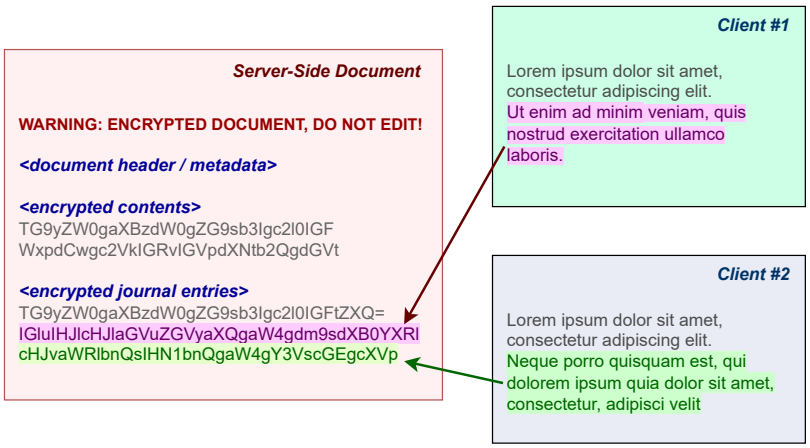


Figure 3.1: Encrypted collaborative document structure.

3.2 SecCollab - Improving Confidentiality for Existing Cloud-based Collaborative Editors

We present a method for providing confidentiality and integrity for these applications while preserving their core feature, that of multiple authors being able to collaboratively edit the same document in real-time, for example, Google Docs.

Thus, there will be two versions of the document: server-side and client-side, illustrated in Figure 3.1. We only use the server-side document as an encrypted storage and synchronization mechanism based on its collaborative protocol. We take each operational transformation made by the user on its decrypted document and append it inside a special, encrypted journal block. The usual operations done that the server will see here are just additions. When the journal gets too large, a client will acquire a mutex (lock) on the document and run a special snapshotting (compaction) operation.

We have implemented a proof of concept addon for the Google Docs service in the form of a browser extension, acting as middleware agent between the web page and the server, intercepting and altering the traffic made by AJAX (Asynchronous JavaScript and XML) such that the server only received encrypted data.

In our implementation, each journal entry is a JSON string (containing transformation metadata) that weighs approximately 40 bytes. Additionally, we employed a AES256-GCM encryption algorithm uses a 96-bit initialization vector plus a 64-bit authentication tag, with a total of $16bytes$. The Base64 encoding also has an overhead given by the formula $ceil(n/3) * 4$ plus a padding (1-4 bytes), so the resulting size of an encrypted journal block is $ceil((41 + 16)/3) * 4 + pad = 76 + pad = 80$ bytes. Our snapshotting process limits the maximum size of the journal to $33.6KB$.

Chapter 4

Trusted Application Development

In this chapter, we describe the application development challenges introduced by trusted execution: requirements for trusted versus untrusted API calls, abstractions for cross-platform TEE software, and state-of-the-art compatibility layers for running normal, unmodified programs inside TEEs.

4.1 Development Model

Normally, existing applications that wish to use trusted execution technologies will require a re-design, i.e., for their code to be split in two: a trusted, security-critical component (to be run inside the secure environment) and an untrusted part (for unsecure execution inside a rich OS). Moreover, each TEE platform uses a different architecture and API for the development of trusted applications. Thus, more development effort is required from vendors wishing to support multiple TEEs, which translates to increased costs. Reusable abstractions (e.g., in the form of libraries) can be used to alleviate this, although they can potentially increase TCB size.

Another problem is related to the additional design requirements of trusted software. A typical application uses the OS kernel system call interface for accessing the filesystem, communicating with other processes or over the network, reading input from the users and displaying results on the screen etc. A secure application might require some of these features and will either need them to be implemented by the trusted environment, or for some of them, relayed to the OS for execution.

Solutions exist, usually by adding an OS compatibility layer inside the TEE to delegate the system calls either to the trusted platform's services, or even to the untrusted OS (and securing the interaction by other means), as we present in the next section.

4.2 Trusted Development Frameworks

Frameworks for Trusted Execution Environment development are hot research topic. This sections reviews such solutions grouped by their targeted platform. We also present several cross-TEE abstraction layers that help developers target multiple platforms at once.

The first one, ARM TrustZone, requires a special trusted firmware to be developed (bootloader, special Monitor handler and trusted microkernel). Frameworks such as PrivateZone [11] allow developers to easily split the program and run parts of their applications

inside the secure world, while other works focus on bringing specialized programming language support for TEEs (e.g., .NET Framework, Rust).

TrustShadow [12] describes a method to run unmodified Linux binaries inside the Secure World using a transparent system call forwarding scheme: applications inside the TEE have their requests proxied over to the rich OS, executed, then results copied back. Our work, *HiddenApp*, also does system call proxying for running Linux programs inside the trusted environment. The main difference is that we use an untrusted user-space process (wrapper) as gateway for the calls in a similar way to Intel SGX's OCall approach.

For Intel SGX user-space enclaves, the developers must specifically write a library with the code to run inside the TEE context. Several SDKs were built to help this (OpenSGX, Intel SGX SDK). Running unmodified applications inside Intel SGX enclaves is possible by employing library OSes such as Haven [13] for Windows applications, SCONE [14] for Linux containers like Docker and Graphene-SGX [15], which runs unmodified POSIX binaries. These allow easily porting applications to enclaves at the expense of increased TCB complexity.

4.3 HiddenApp - Securing Linux Applications Using ARM TrustZone

HiddenApp is our solution for easily porting existing programs to an ARM TrustZone [3] trusted environment. Recall that TrustZone's Secure World also has separate privilege levels for user / kernel modes. Thus, we developed a microkernel providing secure services while also functioning as an OS abstraction layer, intercepting the system calls from the TEE, forwarding them to the normal (Rich) OS for execution then returning their results back to the trusted application. Note that the application's source code does not require any changes, offering an easy way for protecting it in the face of an untrusted OS and minimizing the TCB.

Our solution uses multiple components. When the system boots, a secure loader will properly configure the memory, interrupt controller and other TrustZone aware peripherals (e.g., serial console, touch input or display) and install a Monitor Mode handler (for Secure Monitor Calls – `SMC`, switching context between the secure / normal worlds). Note that, with ARM TrustZone, a segment of the SoC's physical memory will be reserved for the Secure World and will be inaccessible to the normal OS. The trusted firmware may block several interrupts from reaching the normal OS (e.g., to intermediate access for trusted devices). After that, both the **Trusted Microkernel** and the **Rich Operating System** are launched into their respective execution contexts. The normal OS (a GNU/Linux kernel) is passed control and waits for trusted applications to be run by the user.

Both the *Trusted Microkernel* and the *TrustZone Communication Kernel Module* will need to marshal / unmarshal the system call data using a custom packet structure. Thus, they require abstraction code for all the available Rich OS service routines: their parameters' order, data types, buffer directions / lengths and their overall behavior (i.e., the kernel's

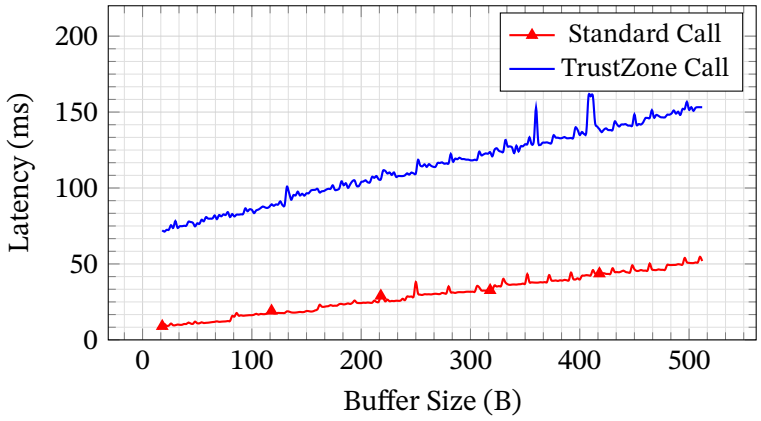


Figure 4.1: Write syscall durations for various buffer sizes [16].

userspace API). Each system call must also have proper checks as to prevent unauthorized access (e.g., Iago attacks) or buffer overflow vulnerabilities. Custom trusted features may also be implemented at this layer, e.g. transparent filesystem encryption.

We implemented and tested our solution on a NXP i.MX53 development board powered by Linux v2.6.35, using a modified U-Boot as bootloader for doing TrustZone platform initialization and booting the OSes. Our microkernel was designed to be as small as possible. This resulted in a TCB of $\approx$ 2000 lines of code (C + assembly), excluding the crypto library (LibTomCrypt) used for RSA-based digital signature verification with a large footprint (30 *KB* as binary). We validated it by testing simple programs requiring OS interaction such as filesystem read/write, a telnet and a ssh client.

The syscall latency is shown in Table 4.1. An obvious issue is the large overhead of $\approx$ 49 *ms* between normal and proxied system calls, mainly due to the many Supervisor and Monitor mode context switches made in this process. We can see that the time is amortized when the system call's execution takes longer, as is the case with the `write` routine. Although the performance difference is quite large, we emphasize the security improvements from protecting the applications' sensitive data against powerful attackers and consider the costs as acceptable. Speed optimizations might still be possible, though we leave them as future work.

Table 4.1: Average latency for tested system calls [16].

Name	Standard (ms)	TrustZone (ms)	Overhead (%)
getpid	0.58	49.33	8325 %
socket	12.6	61.5	384 %
write (256B)	29.60	114.27	386 %
write (512B)	51.92	153.19	295 %

Chapter 5

Embedded Devices in Trusted Infrastructures

Traditionally, embedded devices usually lack modern security due to their constrained capabilities (processing power, memory and firmware code size). Cryptography is often necessary for ensuring the information security of such embedded systems, but the implementation cost is often regarded as being high. Because of the demanding computational power and memory that these algorithms require, and the limited resources available from the hardware, developers have previously used inadequate security practices or even skipped them entirely. Nowadays, as technology continuously improves, we have faster, smaller and more power efficient processors, up to the point that the costs of adding security to embedded applications have become low enough to be feasible, justified even for low-end products.

In this chapter, we test several modern microcontroller platforms, benchmarking various cryptographic algorithms and libraries over multiple devices to help choose the most appropriate product for a cost-efficient, but secure design. Our work is used later in this thesis for the design of embedded devices used to augment trusted infrastructures with missing features (such Trusted I/O and trusted industrial systems firewall).

5.1 Evaluation of cryptographic primitives on modern microcontroller platforms

We evaluated the performance of multiple open-source libraries marketed for embedded use, selecting popular cryptographic algorithms (both symmetric and asymmetric encryption, hashing and authentication) on several low-cost ARM Cortex-M processors. We implemented a modular firmware in C that allowed us to test them all, report and compare their results.

The overall performance comparison of the symmetric algorithms is shown in Figure 5.1.

The last algorithm to be tested, the asymmetric cipher RSA (Table 5.1) had the hardest impact on the embedded systems. No implementation could run on EFM32ZG because of the program size exceeding its flash memory ($40KB$ versus $32KB$) regardless of any optimizations we tried. The other two microcontrollers had a single 128 bytes operation take a lot more cycles to complete, as expected from the heavyweight RSA cryptosystem.

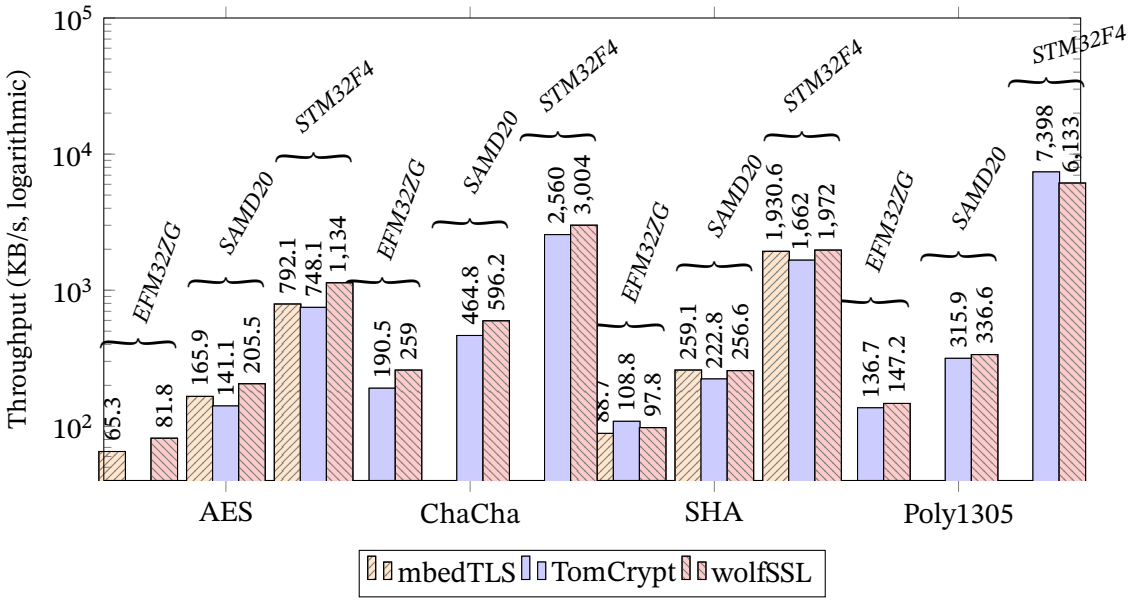


Figure 5.1: Throughtput comparison of AES, ChaCha, SHA and Poly1305 on all platforms

Platform	Library	Prog.Size (Bytes)	Memory (Bytes)	Decrypt (ms)	Encrypt (ms)
SAMD20	mbedTLS	46160	5412	830.6	44.1
SAMD20	TomCrypt	42696	8536	3335.4	340.3
SAMD20	wolfSSL	44720	3272	3699.5	364.3
STM32F4	mbedTLS	44628	6978	88.6	4.2
STM32F4	TomCrypt	41256	10112	274.0	22.8
STM32F4	wolfSSL	42360	4856	348.4	28.8

Table 5.1: Results for the asymmetric RSA cipher (1024 bit).

Finally, the power requirements are presented in Figure 5.2. The current consumption was essentially the same for all implementations on a platform, regardless of the algorithm executing, so the table only includes the combined results.

5.2 Energy Considerations Regarding Transport Layer Security in Wireless IoT Devices

Commercial Internet of Things (IoT) appliances are expected to protect their users' privacy and to prevent control by remote attackers, but usually fail to do so due to insufficient security measures or improper implementations. Many IoT specific application protocols that are optimized for low complexity and low resource usage, such as CoAP or MQTT. These protocols rely on other layers to provide security, such as a secure communication channel ensured by Transport Layer Security (TLS) which is seldom employed due to power

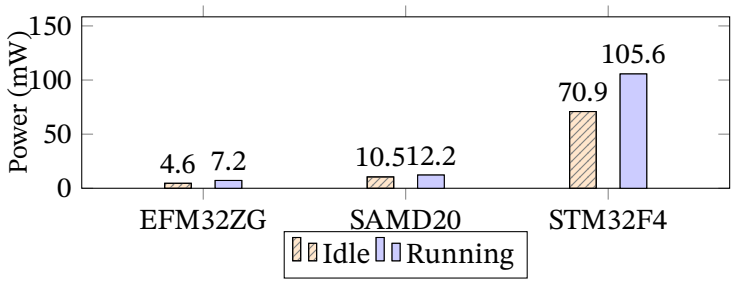


Figure 5.2: Power consumption (idle versus execution).

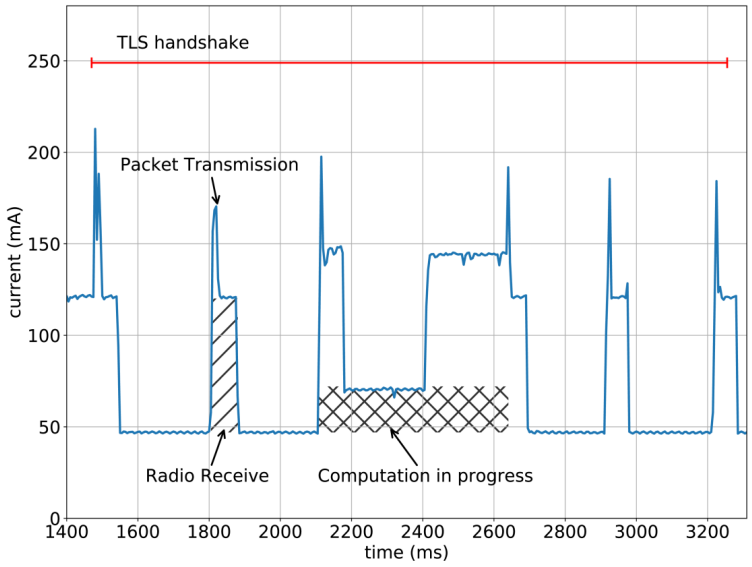


Figure 5.3: TLS Handshake - Current Waveform, DMM Capture [17]

consumption considerations.

Our work measures the energy impact of using the Transport Layer Security protocol in WiFi-enabled IoT devices (we tested an Espressif ESP32 development board) and shows that, under real-world conditions, is actually of low concern.

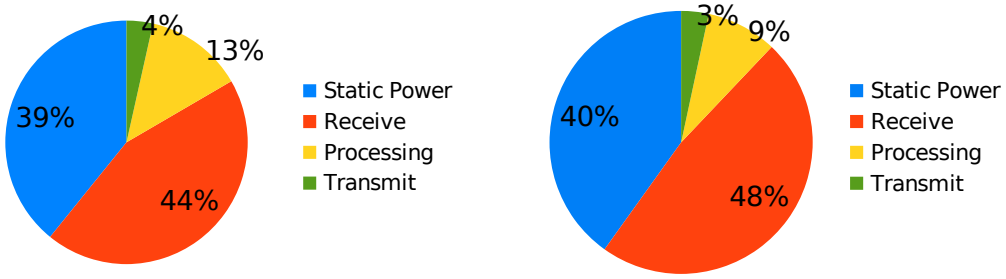
We used a Digital Multimeter interfaced with a computer to measure the current consumption of the device under test, together with an oscilloscope connected in parallel for capturing more accurate peaks (due) and having a secondary channel connected to a GPIO, signalling the various states of the protocol (illustrated in Figure 5.3).

Using the above methodology, the device's performance (processing time and current usage) is evaluated using multiple scenarios. First, the baseline current values will be measured for different basic states and operating modes: idle, a NOP loop, and the targeted algorithms in a synthetic environment (i.e., with no network activity, radio off). These baseline values are, then, matched onto the complex waveforms obtained when running a

real TLS session.

The energy consumption contributions were plotted in Figure 5.4. We see that the radio (in receive mode) only requires about half of the total power, the rest (static power) is used by the microcontroller just being active.

Regarding TLS overhead, the handshake operation is significantly more costly than application data exchange, which is performed using a symmetric cipher. In applications that exchange large volumes of data, the one-time character of the handshake makes its overall contribution negligible. In applications that transmit data occasionally, it may be advantageous to disconnect from the wireless network and place the device in a low-power sleep state, but the connection needs to be re-established, forcing the significant handshake overhead.



(a) TLS Handshake

(b) TLS Receive

Figure 5.4: TLS Energy Contributions [17]

Chapter 6

Trusted I/O Path

6.1 Problem Overview

The Operating System is traditionally responsible for interfacing with the external world (e.g., network, storage, peripherals). Programs running inside a Trusted Execution Environment may freely make use of OS services, but they are not to be trusted. However, some use cases might require secure access to specific hardware components. For example, the keyboard is often used as method of authentication (e.g., via passphrases). Unfortunately, some popular platforms (e.g., PCs) lack integrated means of establishing trusted communication paths with its peripherals, which we will discuss in this chapter.

Many security-critical applications require interaction with the peripherals (e.g., keyboard, display, touchscreen) or some other devices (e.g., industrial equipment connected over serial adapters). These applications would greatly benefit from being isolated inside a Trusted Execution Environment, but the usual way they interact with the hardware is by making use of untrusted Operating System services (via its device drivers). To protect against this, either access to the specific hardware peripherals needs to be denied from the OS, or a trusted communication channel must be established with the application TEE such that a malicious Man-in-the-Middle kernel would be unable to interfere. This is defined as the *Trusted I/O Path* problem [18], and there are multiple approaches for solving it depending on the peripheral device's class and available platform features.

6.2 Systematization of Trusted I/O Solutions for TEEs

In this section, multiple works in the Trusted I/O Path field will be presented, highlighting their novelty concepts, notable differences and improvements. Note that, to each of the articles, a single-word alias has been given, which will be used through the rest of the paper for easy identification.

As presented in Table 6.1, the available trusted path solutions are diverse, with varying application classes and platform requirements. We used multilateral classification by: peripheral type (Human Input Device / Display / others); the I/O isolation mechanisms used (virtualization-based - *Virt*, chipset-based access control - *Chip*, external device - *Ext*); interface / protocol: USB / Bluetooth / Network; for virtualization-based approaches, their implementation method: MMIO (memory mapped I/O) / device driver virtualization.

Note that the half-circle denotes a partial implementation of the feature; for HID, it means

Name	Isolation	Interface	HID	Display	Others	Target TEE	+Hypervisor	+Chip/firmware	+Ext. Dev	TCB LoC
			Peripherals				TCB Additions			
ZTIC [19]	Ext	USB	●	◐	○	*Remote*	✗	✗	✓	$\approx$ 110KB [1]
Bumpy [20]	Ext	USB	●	○	○	Flicker	✗	✗	✓	$\approx$ 8.5k
Bumpy Display	Ext	Network	○	◐	○		✗	✗	✓	$\approx$ 10k
UTP [21]	Virt	Driver	●	◐	○	Flicker	✓	✗	✗	$\approx$ 2.3k
VTP x86 [18]	Virt	MMIO	●	◐	●	TrustVisor	✓	✗	✗	$\approx$ 15k
Intel PAVP [22]	Chip	GPU	○	●	○	DRTM, SGX	✗	✓	✗	n/a
TrustUI [23]	Virt	Driver	◐	◐	○	TrustZone	✓	✗	✗	$\approx$ 10k
Wimpy [24]	Virt	MMIO	●	○	●	DRTM	✓	✗	✗	$\approx$ 3.5k
GSK [25]	Virt	GPU MMIO	○	●	○	TrustVisor	✓	✗	✗	$\approx$ 35k
SGXIO [26]	Virt	Driver	●	○	○	SGX	✓	✗	✗	n/a
BASTION-SGX [27]	Chip	Bluetooth HCI	●	○	●	SGX	✗	✓	✗	n/a
ProximiTEE [28]	Ext	USB	●	○	○	SGX	✗	✗	✓	$\approx$ 5k
SecDisplay [29]	Virt	USB	●	◐	○	TrustZone	✓	✗	✗	$\approx$ 2k
TIO [30]	Ext	USB	●	○	●	SGX	✗	✗	✓	$\approx$ 27k [2]
Aurora [31]	SMM Virt	MMIO	●	○	●	SGX	✗	✓	✗	$\approx$ 3.3k + 696KB [3]

Table 6.1: Comparison of Trusted Path Solutions.

[1] only binary size given [2] counts firmware, enclave framework & full asymmetric crypto lib [3] hypervisor (LoC) + enclave library (compiled binary)

only a subset on input devices are usable; for display, it means text-only output. The additional TCB components are given (including the added device / chip module, besides from the CPU platform).

A first observation is that implementing trusted graphical display is a hard problem: many solutions only worked for partial, text-mode output either using an external LCD or secure console output (using BIOS-like text mode switching).

We note that, for the Trusted Computing Base - a desirable comparison metric, the size values were taken as-is from the papers and are unreliable for this purpose because of differences in measurement methodologies (e.g., lines of code vs binary sizes, different supported devices / feature sets, target platforms incurring framework overhead, unoptimized cryptographic libraries used).

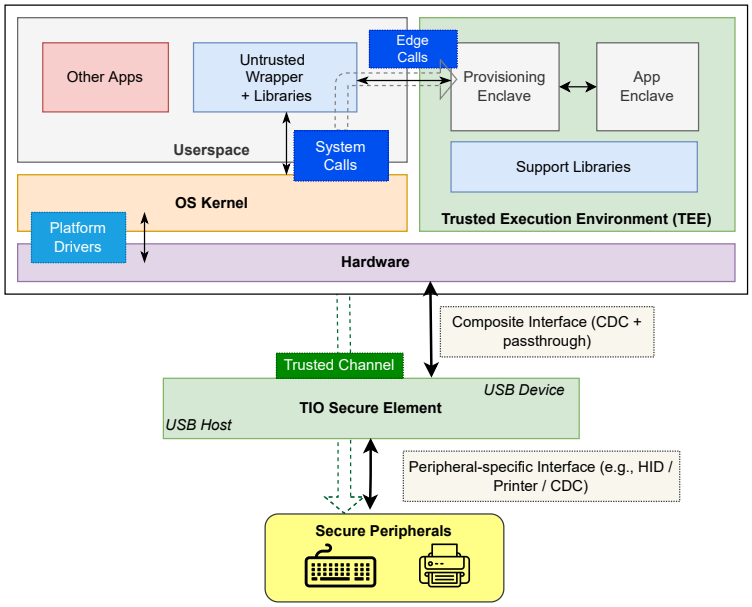
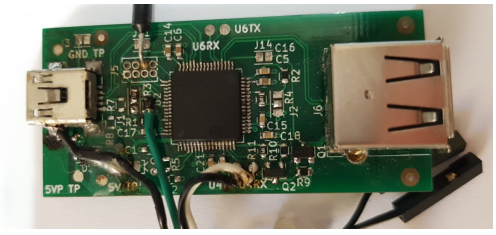


Figure 6.1: TIO System Architecture.

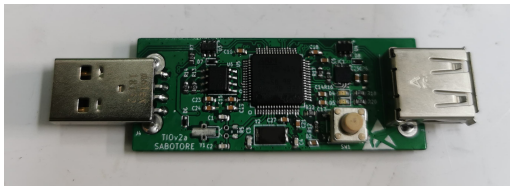
6.3 TIO - Secure Input/Output for Intel SGX

TIO consists of a custom hardware dongle with two USB ports: a host interface for connecting generic peripherals, and a separate USB device interface for interacting with the SGX-enabled PC. Thus, the TIO secure element acts as a transparent I/O gateway, establishing a secure channel between the enclave and an otherwise unaware peripheral, for exchanging USB packets. The overall system design is illustrated in Figure 6.1.

When establishing the secure channel between an enclave and the TIO module, we use a proper authentication protocol that protects the user against man-in-the-middle attacks by mutual authentication of the involved parties. We solve this by introducing a one-time setup prerequisite: the trusted device needs to be paired to the user's machine beforehand (by booting a read-only Linux OS stored inside the flash memory of the dongle).



(a) v1 (with debugging wires attached)



(b) v2 (cleaner, Type A device connector)

Figure 6.2: TIO hardware prototypes.

Our TIO prototypes use a low-cost STM32F405 ARM Cortex-M4 microcontroller running at 168MHz with 192KB SRAM and 1MB flash. It also includes 2xUSB OTG interfaces, one

used for connection with the PC, and the other one configured as USB host for trusted peripherals. The dongle-like device was packaged as a small-factor printed circuit board, with connectors and auxiliary components, as presented in Figure 6.2.

Table 6.2: TCB size measurements for the microcontroller firmware and the enclave, split components.

(a) Microcontroller Firmware

Module	**LoC**	**ObjCode (KB)**
mbedTLS	15018	44.1
STM32 SDK	8277	18.1
Firmware code	3215	11.2

(b) Device Provisioning Enclave

Module	**LoC**	**ObjCode (KB)**
IPP Crypto	22955	254
SGX tRTS	9742	111
DPE code	687	11.1

The TCB size results are presented in Table 6.2a for the *microcontroller firmware*, and in Table 6.2b for the *Enclave*. We observe that the enclave has a bigger total footprint compared to the embedded code due to the large Intel SGX framework employed.

Table 6.3: Performance measurements.

Handshake Time	1165 ms
HID Report Latency Overhead (RTT)	1.3 ms

In Table 6.3, we measured the time taken for the enclave to do the initial Diffie-Hellman handshake with the microcontroller and the latency overhead of input reports from a HID peripheral (a USB keyboard).

Our implementation currently understands the USB HID and Printer class protocols. Our solution can be extended to cover other peripheral classes by implementing the appropriate device class inside the enclave (for certain applications, the developer must also enhance the microcontroller's firmware for class-specific I/O state management / filtering, or for adding activity indication).

Chapter 7

Trusted Industrial Infrastructure

Any use of connected computers opens the issue of security. This is also the case for the industrial systems used for efficiently automating the processes of a factory or plant, since it requires their embedded devices, controllers and supervisory PCs to communicate with each other to ensure a proper execution logic, present a friendly interface to the users for monitoring etc. Since such systems are able to control physical machinery like electric motors or valves, the damage potential in case of a cyber-attack could have devastating consequences.

In this chapter, we explore using Trusted Execution technologies in an effort to improve the security of sensitive applications by separating them from a potentially vulnerable rich operating system such that, if exploited, the attacker's reach will be limited.

Our work proposes an architecture employing TEEs and a firewall device to isolate the security-sensitive industrial control network from general-purpose systems (e.g., operator PCs, remote servers) which may be exposed to cyber-attacks from external sources (e.g., Internet, USB sticks), while still providing the means for issuing both normal and exceptional (emergency stop) commands to the automation controllers securely. To this end, we enhance the most popular industrial electronics communication protocol, Modbus, with cryptographic authentication features and design an affordable firewall device to make it easy for legacy equipment to transition to a secured network. Since realtime requirements are mandated, we demonstrate that even low-power, off the shelf microcontrollers can handle the cryptographic computations required for a secure protocol with acceptable performance.

Our architecture, illustrated in Figure 7.1, introduces an authenticated communication channel between the control equipment and trusted applications running on untrusted devices (on the corporate / management network which is, presumably, connected to the Internet). We used a low cost, low power, microcontroller-based firewall as model for integrating a network of legacy devices over a serial network, though each of the controllers may implement the authenticated Modbus protocol separately due to the low hardware requirements of our solution. Because of this, we further abstract the individual industrial devices and consider our gateway as the sole control system within our protocol.

We modify Modbus in a compatible way, adding a cryptographic authentication protocol by using a custom function code and encapsulating everything else as data, so any protocol translation devices in the way (such as Modbus TCP to Modbus RTU) will continue to work as expected.

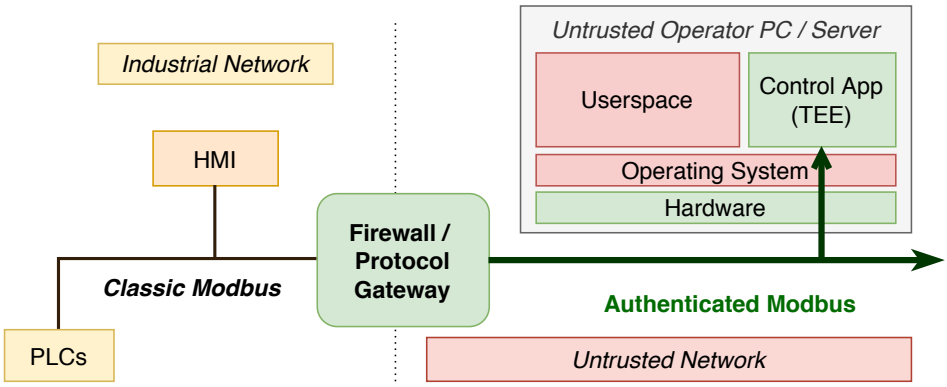


Figure 7.1: Trusted ICS Architecture.

We implemented a firewall prototype using an Olimex STM32F4 microcontroller development kit. The MCU has a maximum frequency of 168MHz, 192KB of RAM memory and 1MB flash storage. Two of the board's UARTs (serial interfaces) were wired to a Raspberry Pi and a laptop simulating the industrial control devices / master using Python.

We used a logic analyzer (Figure 7.2) to determine the performance / latency overhead introduced by our modified protocol (since Modbus RTU's has strict timings). The total MCU run time of the authenticated Diffie Hellman operations is of $\approx$ 1100 *ms*, though it is split among 2 requests.

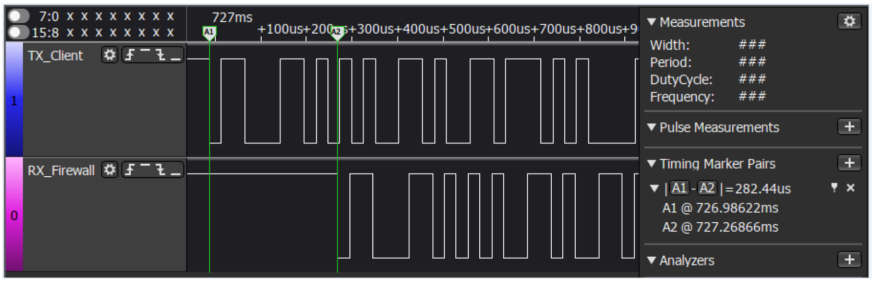


Figure 7.2: Logic Analyzer Output

In normal operation conditions, the symmetric MAC computation latency introduced is $280uS/block$ ($1\ block = 16\ bytes$). For calculating Modbus's maximum silence period: $t_{silence} = \frac{3.5*10}{baudrate}$, which, for a baudrate of 115200 we obtain: 303 uS. There is also a fixed PDU size overhead of 33 *bytes*, which would increase the transmission duration (especially on low baud rates), but otherwise it won't affect any of the protocol's requirements.

Chapter 8

Conclusions

Our thesis focused on Trusted Execution Environments (TEE), a technology providing isolated hardware areas which can be used to protect sensitive applications against more privileged malware and by reducing the complexity of the Trusted Computing Base (TCB), the set of security-critical components in a system.

We explored several specific research issues: cloud trustworthiness, trusted software architecture & engineering challenges, the feasibility of embedded devices in trusted infrastructures, ensuring a trusted I/O path for hardware peripherals and hardening the security of industrial devices with the help of TEEs.

8.1 Thesis Summary

In Chapter 2, we described the evolution of trusted computing concepts leading to the emergence of practical trusted execution environments. We presented the various isolation technologies available in commodity hardware (for both embedded / mobile devices: ARM TrustZone, and the general computing chips: Intel's Trusted Execution Technology, its successor, Software Guard Extensions and AMD's Secure Encrypted Virtualization), along with related scientific publications.

In Chapter 3, we begin by reviewing the cloud trustworthiness problem and approaches for protecting the users' data on untrusted servers. We argue that new solutions are enabled by the novel trusted technologies and their current / upcoming availability in commercial offerings from top vendors (Amazon AWS, Microsoft Azure, Google Cloud), a move that can increase the trustworthiness of cloud services. We develop *SecCollab*, our approach for ensuring privacy in cloud-based document editing systems. It takes the form of a browser extension and leverages the existing differential synchronization protocol used by collaborative applications, implementing an encrypted journal on top of it in order to preserve both the confidentiality of the users' documents and their real-time editing features.

Next, in Chapter 4, we presented how the trusted execution concept requires changes in the software development paradigms for trusted applications. As the Operating System is now considered untrusted, programs relying on its services (e.g., filesystem, networking, hardware I/O access) must protect their secrets using additional means (cryptography or other trusted platform services). In Section 4.2, we discussed several research works aimed at minimizing the development effort: running programs inside TEEs with little to no modifications, or frameworks for building cross-platform trusted applications. We also

described our approach, HiddenApp (4.3), for enabling trusted execution of existing Linux applications inside an ARM TrustZone TEE. For this, we developed a microkernel that can run unaltered programs inside the Secure World by intercepting their system calls and forwarding them to the Rich OS for processing.

Chapter 5 showed that modern microcontrollers can successfully run cryptographic code with reasonable performance, mostly depending on the power requirements of the application. Multiple open-source cryptographic libraries were evaluated to run on small devices where storage space and memory are scarce, and has been shown that the energy consumption due to cryptographic processing is negligible. We later used this as a building block for designing embedded devices to enhance the security of trusted applications.

With Chapter 6, we move to the Trusted I/O Path problem: ensuring secure communication between Trusted Execution Environments and hardware peripherals in the face of an untrusted OS. In Section 6.2, we presented a systematization of the existing state-of-the-art trusted path solutions for various TEE platforms (Intel TxT, ARM TrustZone, Intel SGX) and devices (categorized by type: HID input / display output; by interface: memory-mapped I/O, GPU, USB, Bluetooth etc.). In 6.3, we developed *TIO*, a hardware-based approach for establishing a secure I/O pathway between generic USB devices and Intel SGX-based TEEs. Our embedded device is a small and practical way to enhance the trustworthiness of applications requiring secure user interaction, which we demonstrate by protecting keyboard input and securely printing PDF documents.

Finally, in Chapter 7, we described an architecture for securing industrial control systems. Our solution is based on a low-cost firewall device situated at the boundary between the sensitive control network and the untrusted management / corporate networks or even behind each cyber-physical unit. A trusted application (residing inside an enclave on the operator's PC) establishes a trusted I/O channel with the embedded firewalls and signs each request with a shared authentication key, which the device can then use to filter out any malicious commands, thus preventing sabotage even from privileged attackers. We also retrofitted a popular industrial protocol (Modbus) with cryptographic integrity fields for ensuring the authenticity of the packets sent within legacy networks.

8.2 Contributions

In our thesis, we made several original contributions to solve some of the current issues that might make the adoption of trusted execution technologies difficult: trusted application development and trusted I/O path. We also propose solutions to improve the security in all popular application domains: cloud, personal, embedded and industrial computing.

1. We gave a comprehensive background on the Trusted Computing history, TEE technologies currently available on commodity CPUs, along with reviewing numerous state-of-the-art works related to this field.

2. We designed SecCollab [32], a method for ensuring the confidentiality of online, web-

based collaborative document editing applications by using a browser extension to add client-side encryption to differential synchronization protocols.

3. We presented the application development challenges of targeting isolated environments and implemented HiddenApp [16], a solution for running unmodified Linux applications inside ARM TrustZone's Secure World.

4. We tested the cryptographic performance of multiple embedded devices and software libraries [33, 17] with the aim of using them as trusted devices.

5. We approached the problem of securing the interaction of TEEs with input / output peripherals from untrusted Operating Systems, proposing TIO [30], a low-cost hardware-based solution for Intel SGX enclaves to safely communicate with other USB devices (e.g., keyboard, printers). We also systematized the other Trusted I/O Path solutions, comparing their usability and TCB sizes.

6. We proposed an architecture for improving the security of cyber-physical (industrial) systems [34] with the aid of Trusted Execution Environments and custom, low-cost embedded devices, enhancing the traditionally insecure Modbus protocol to do cryptographic authentication of sensitive control commands or sensor data.

8.3 Future Work

Finally, we state future directions we would like to pursue or see accomplished in the trusted technologies field. First, we argue that cyber-security of general purpose computing would greatly benefit if vendors came together with a more open platform for trusted execution, so developers could rely on standardized set of features (e.g., edge call APIs, remote attestation, trusted I/O for some basic peripherals), increasing adoption of the trusted execution solutions.

We also plan to further develop our embedded TIO USB device to enhance performance, support to other classes of peripherals (e.g., USB Hub) and integrate it to desktop applications requiring enhanced secrets protection (password managers, certificate storage, ssh etc.). We would also like to explore building a portable Trusted Execution Environment as an on-the-go security appliance (using an embedded application CPU like the Raspberry Pi's).

8.4 List of publications

1. Dumitru C. Trancă, **Florin Stancu**, Răzvan Rughiniș and Daniel Rosner, *"SiloSense: ZigBee-based wireless measurement system architecture for agriculture parameter monitoring"*, 2017 4th International Conference on Control, Decision and Information Technologies (CoDIT), Barcelona, pp. 0330-0335, 2017 (IEEE),
DOI: 10.1109/CoDIT.2017.8102613, WOS: 000450826500057.

2. **Florin Stancu**, Mihai Chiroiu and Răzvan Rughiniș, *"SecCollab - Improving Confidentiality for Existing Cloud-Based Collaborative Editors"*, 2017 21st International Conference on Control Systems and Computer Science (CSCS), Bucharest, pp. 324-331, 2017 (IEEE), DOI: 10.1109/CSCS.2017.51, WOS: 000449004400044.

3. Veronica Velciu, **Florin Stancu** and Mihai Chiroiu, *"HiddenApp - Securing Linux Applications Using ARM TrustZone"*, Innovative Security Solutions for Information Technology and Communications (SECITC), 2018 Lecture Notes in Computer Science, vol 11359, 2018 (Springer, Cham), DOI: 10.1007/978-3-030-12942-2_5.

4. **Florin Stancu**, Dumitru C. Trancă, Mihai Chiroiu and Răzvan Rughiniș, *"Evaluation of cryptographic primitives on modern microcontroller platforms"*, 2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet), Cluj-Napoca, pp. 1-6, 2018 (IEEE), DOI: 10.1109/ROEDUNET.2018.8514127, WOS: 000517570500005.

5. Daniel Rosner, Cristiana Trifu, Dumitru C. Trancă, Iuliu Vasilescu and **Florin Stancu**, *"Magnetic Field Sensor for UAV Power Line Acquisition and Tracking"*, 2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet), Cluj-Napoca, pp. 1-5, 2018 (IEEE), DOI: 10.1109/ROEDUNET.2018.8514123, WOS: 000517570500002.

6. Răzvan Tataroiu, **Florin Stancu** and Dumitru C. Trancă, *"Energy Considerations Regarding Transport Layer Security in Wireless IoT Devices"*, 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, pp. 337-341, 2019 (IEEE), DOI: 10.1109/CSCS.2019.00060, WOS: 000491270300053.

7. **Florin Stancu**, Dumitru C. Trancă and Mihai Chiroiu, *"TIO - Secure Input/Output for Intel SGX Enclaves"*, 2019 International Workshop on Secure Internet of Things (SIOT), 2019 (IEEE), DOI: 10.1109/SIOT48044.2019.9637105.

8. **Florin Stancu**, Răzvan Rughiniș, Dumitru C. Trancă and Ioana Popescu, *"Trusted Industrial Modbus Firewall for Critical Infrastructure Systems"*, 2020 RoEduNet (19th RoEduNet Conference: Networking in Education and Research), 2020 (IEEE), DOI: 10.1109/RoEduNet51892.2020.9324884, WOS: 000654265900033.

9. **Florin Stancu**, Alexandru Mircea, Răzvan Rughiniș, Mihai Chiroiu, *"Systematization of Trusted I/O solutions for Isolated Eecution Environments"*, accepted for publication at U.P.B. Scientific Bulletin, Series C, Bucharest, Romania, 2022 (Journal).

Bibliography

[1] M. Larabel, Phoronix, "The Linux Kernel Enters 2020 At 27.8 Million Lines In Git," https://www.phoronix.com/scan.php?page=news_item&px=Linux-Git-Stats-EOY2019, January 2020.

[2] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security & Privacy*, vol. 12, no. 4, pp. 29–37, 2014.

[3] ARM Holdings, "ARM TrustZone Security Extensions," https://developer.arm.com/technologies/trustzone.

[4] Intel, "Intel SGX Software Guard Extensions," https://software.intel.com/en-us/sgx.

[5] D. Kaplan, J. Powell, and T. Woller, "AMD Memory Encryption," *White paper*, 2016.

[6] J. Criswell, N. Dautenhahn, and V. Adve, "Virtual ghost: Protecting applications from hostile operating systems," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 81–96, 2014.

[7] U. Lee and C. Park, "SofTEE: Software-based trusted execution environment for user applications," *IEEE Access*, vol. 8, pp. 121 874–121 888, 2020.

[8] W. Futral and J. Greene, "Fundamental principles of intel® txt," in *Intel® Trusted Execution Technology for Server Platforms*. Springer, 2013, pp. 15–36.

[9] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP Journal on Information Security*, vol. 2007, pp. 1–10, 2007.

[10] F. Y. Rashid, "The rise of confidential computing: Big tech companies are adopting a new security model to protect data while it's in use-[news]," *IEEE Spectrum*, vol. 57, no. 6, pp. 8–9, 2020.

[11] J. Jang, C. Choi, J. Lee, N. Kwak, S. Lee, Y. Choi, and B. B. Kang, "Privatezone: Providing a private execution environment using arm trustzone," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 5, pp. 797–810, 2018.

[12] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "TrustShadow: Secure execution of unmodified applications with ARM trustzone," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 488–501.

[13] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with Haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, pp. 1–26, 2015.

[14] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell *et al.*, "{SCONE}: Secure linux containers with intel {SGX}," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 689–703.

[15] C.-C. Tsai, D. E. Porter, and M. Vij, "{Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.

[16] V. Velciu, F. Stancu, and M. Chiroiu, "Hiddenapp – Securing linux applications using ARM TrustZone," in *International Conference on Security for Information Technology and Communications*. Springer, Cham, 2018, pp. 41–52.

[17] R. Tataroiu, F. A. Stancu, and D.-C. Tranca, "Energy considerations regarding Transport Layer Security in wireless IOT devices," in *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*. IEEE, 2019, pp. 337–341.

[18] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune, "Building verifiable trusted path on commodity x86 computers," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 616–630.

[19] T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch, "The Zurich Trusted Information Channel–an efficient defence against man-in-the-middle and malicious software attacks," in *International Conference on Trusted Computing*. Springer, 2008, pp. 75–91.

[20] J. M. McCune, "Safe passage for passwords and other sensitive data," in *Proceedings of the Network and Distributed System Security Symposium, 2009*, 2009.

[21] A. Filyanov, J. M. McCuney, A.-R. Sadeghiz, and M. Winandy, "Uni-directional trusted path: Transaction confirmation on just one device," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2011, pp. 1–12.

[22] X. Ruan, *Platform Embedded Security Technology Revealed*. Springer Nature, 2014.

[23] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C.-K. Chu, and T. Li, "Building trusted path on untrusted device drivers for mobile devices," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, 2014, pp. 1–7.

[24] Z. Zhou, M. Yu, and V. D. Gligor, "Dancing with giants: Wimpy kernels for on-demand isolated i/o," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 308–323.

[25] M. Yu, V. D. Gligor, and Z. Zhou, "Trusted display on untrusted commodity platforms," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 989–1003.

[26] S. Weiser and M. Werner, "SGXIO: Generic trusted I/O path for Intel SGX," in *Proceedings of the seventh ACM on conference on data and application security and privacy*, 2017, pp. 261–268.

[27] T. Peters, R. Lal, S. Varadarajan, P. Pappachan, and D. Kotz, "BASTION-SGX: Bluetooth and architectural support for trusted I/O on SGX," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018, pp. 1–9.

[28] A. Dhar, I. Puddu, K. Kostiainen, and S. Capkun, "ProximiTEE: Hardened SGX attestation by proximity verification," in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, 2020, pp. 5–16.

[29] J. Cui, Y. Zhang, Z. Cai, A. Liu, and Y. Li, "Securing display path for security-sensitive applications on mobile devices," *Computers, Materials and Continua*, vol. 55, no. 1, p. 17, 2018.

[30] D. C. T. F. A. Stancu and M. Chiroiu, "TIO – Secure Input/Output for Intel SGX Enclaves," in *International Workshop on Secure Internet of Things (SIOT)*, 2019.

[31] H. Liang, M. Li, Y. Chen, L. Jiang, Z. Xie, and T. Yang, "Establishing trusted I/O paths for SGX client systems with Aurora," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1589–1600, 2019.

[32] F. A. Stancu, M. Chiroiu, and R. Rughinis, "SecCollab – Improving Confidentiality for Existing Cloud-Based Collaborative Editors," in *2017 21st International Conference on Control Systems and Computer Science (CSCS)*. IEEE, 2017, pp. 324–331.

[33] F. A. Stancu, C. D. Trancă, M. D. Chiroiu, and R. Rughiniş, "Evaluation of cryptographic primitives on modern microcontroller platforms," in *2018 17th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE, 2018, pp. 1–6.

[34] F. A. Stancu, R. V. Rughinis, C. D. Tranca, and I. L. Popescu, "Trusted Industrial Modbus Firewall for Critical Infrastructure Systems," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE, 2020, pp. 1–5.