Towards a Domain-specific API for Developing and Executing Integration Processes in Trusted Execution Environments *

**Regis Schuch[1], Rafael Z. Frantz[1], Antonia M. Reina Quintero[2], José Bocanegra[3]
Sandro Sawicki[1], Fabricia Roos-Frantz[1], Carlos Molina-Jiménez[4]**

[1]Unijuí University – Ijuí/RS – Brazil

{regis.schuch, rzfrantz, sawicki, frfrantz}@unijui.edu.br

[2]University of Seville – Seville – Spain

reinaqu@us.es

[3]Universidad Distrital Francisco José de Caldas – Bogotá – Colombia

jjbocanegrag@udistrital.edu.co

[4]University of Cambridge – Cambridge – United Kingdom

carlos.molina@cl.cam.ac.uk

Abstract. *We contribute a domain–specific and technology agnostic API for implementing integration processes executed in Trusted Execution Environments to prevent data exfiltration at run–time. We demonstrate its potential with a use case implemented with compartments created on Morello Boards.*

1. Introduction

Integration processes are applications that retrieve data from several remote and independent services and process it locally to implement new services. They are common in smart cities and other domains. Current integration processes protect data in transit but lack mechanisms for preventing data exfiltration at execution–time. A solution is to execute the integration process within a Trusted Execution Environment (TEE) implemented in hardware such as Intel SGX, Amazon Nitro and Morello Board Compartments[1] [Intel 2025, ARM 2025]. These and similar technologies offer APIs that are i) technology specific and ii) general purpose. As a result, TEEs are hard to use in integration processes. Our API can solve the problem, it includes typical operations of integration processes.

2. Functionality of the API in a Typical Scenario

Imagine a store that offers complimentary taxis to customers that spend at least $150. We have implemented this scenario (see Figure 1) with our API as an integration process that securely integrates two services: (i) a *Store Service* that provides data containing customers' phones, addresses and receipts; (ii) a *Taxi Service* that takes bookings. The *Integration Process* is a client of the *Store* and *Taxi* services that acts as servers.

*Research funded by the Co-ordination for the Brazilian Improvement of Higher Education Personnel (CAPES) and the Brazilian National Council for Scientific and Technological Development (CNPq) under grants 311011/2022-5, 309425/2023-9, 402915/2023-2. AETHER-US PID2020-112540RB-C44, ALBA-US TED2021-130355B-C32MICIU/AEI/10.13039/501100011033, "EU NextGenerationEU/PRTR" grants funded Antonia. EPSRC/EP/X015785/1(G115169) grant funded Carlos Molina.

[1]We rule out software based solution like homomorphic encryption because they are still immature.

The interaction is mediated by the Launcher that ensures secure communication, enforces execution environment attestation, and manages cryptographic operations, enabling the secure integration of remote applications operating in conventional environments. The Launcher also manages the Code Repository, retrieving, compiling, and deploying the Integration Process into a secure memory compartment.

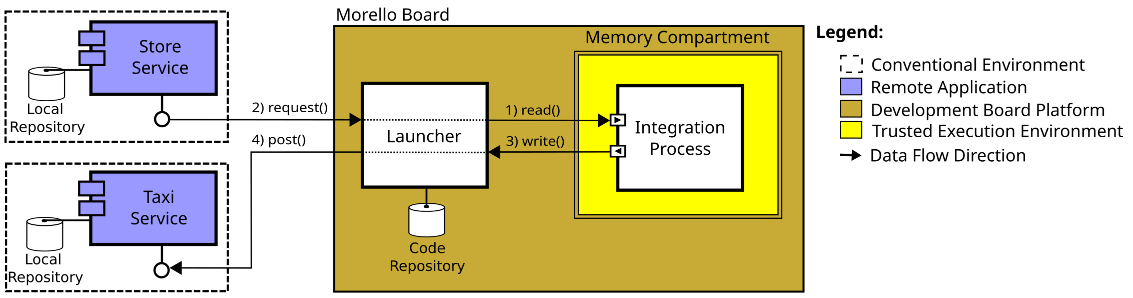

Figure 1. Scenario for implementing the integration process using a TEE.

The workflow begins when the Integration Process issues a `1) read()` request, which the Launcher translates into a `2) request()` and forwards to the Store Service, returning encrypted data. After identifying eligible customers, the Integration Process executes a `3) write()` operation, which the Launcher translates into a `4) post()` to securely transmit customer details to the Taxi Service while ensuring data protection.

The proposed API functions as an abstraction layer, decoupling the domain-specific API (focused on service integration) from the TEE-specific API (responsible for execution, attestation, and security policies). This separation enables integration processes to be deployed across different TEEs, such as Intel SGX, AWS Nitro Enclaves, and Morello Board compartments, without requiring modifications to their core logic. The Launcher ensures that only attested integration processes interact with Remote Applications. If Morello were replaced by Intel SGX, the overall workflow would remain unchanged, requiring only adjustments to attestation mechanisms (e.g., Intel Attestation Service (IAS) versus Morello's compartmentalisation model). The Launcher manages execution by invoking API functions such as `read()`, `request()`, `write()`, and `post()` to securely communicate with Remote Applications. The complete source code for this case study is available at: `https://github.com/gca-research-group/tee-compartimentalisation-study-case`

3. Conclusion and Future Work

We have proposed the use of commodity TEEs to prevent data exfiltration in integration processes. Specifically, we implement an intermediate abstraction layer that decouples a general-purpose programming API from a domain-specific API designed for digital service integration, which abstracts key actions such as secure reading and writing. Future work will focus on extending the case study to additional TEE, including AWS Nitro Enclaves, Intel SGX, AMD SEV, and ARM TrustZone, to evaluate the API's adaptability across different TEEs.

References

ARM (2025). ARM Morello Program. `https://www.arm.com/architecture/cpu/morello`. [online: access in 21-jan-2025].

Intel (2025). Intel® Software Guard Extensions (Intel® SGX). `https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html`. [online: access in 21-jan-2025].