An API for Developing Secure Integration Processes [*]

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

1. Introduction

Integration processes are applications that retrieve data from several remote and independent services, process it locally to implement new services, and potentially transmit data to other integrated services. They are prevalent in smart cities and other domains. Current integration processes protect data in transit but lack mechanisms to prevent data exfiltration during execution. A viable 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 provide APIs that are (i) technology-specific and (ii) general-purpose. Consequently, TEEs are challenging to employ in integration processes. In this paper, we briefly introduce *iDevS*, our API that aims to address this issue by simplifying and encapsulating the common actions involved in integration processes. Integration processes interact with digital services through *receive* and *send* actions to securely retrieve and transmit data. Implementing these actions within TEEs is challenging due to technological variations in execution models, attestation mechanisms, and security policies. The iDevS API abstracts these complexities, standardising secure integration within TEEs. It provides classes that encapsulate these actions, ensuring secure and attested execution. By transparently handling cryptographic and attestation processes, it enables integration processes to execute seamlessly in TEEs such as Intel SGX, AWS Nitro Enclaves, and Morello Board, offering 29 operations across 9 classes. Due to space constraints, this paper focuses on the *send* action; however, the iDevS API also supports the implementation of other relevant actions, such as deployment, compilation, and compartment creation. To validate iDevS, we developed a complete study of a case with the support of our API, which is available on GitHub[2].

[*]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 exclude software-based solutions such as homomorphic encryption, as they remain immature.

[2]https://github.com/gca-research-group/tee-compartimentalisation-study-case

2. Implementing the Send Action

The implementation of the *send* action demands operations from the `Process`, `Service`, and `Launcher` classes of the iDevS API. The `Process` class abstracts operations typically performed by integration processes, such as `getServicePublicKey()` for retrieving a service's public key and `encrypt()` for securing data prior to transmission. The `Service` class encapsulates operations commonly executed by digital services, including `verifyCertificate()` to authenticate the sender and `decrypt()` to restore the original dataset. The `Launcher` class abstracts operations required to mediate secure communication between the integration process and the digital service, such as `lockupService()` to locate the destination service and `getCertificate()` to retrieve the integration process's attestation certificate. Listing 1 presents a pseudocode that shows the use of the iDevS API to encrypt, attest, and validate data during the implementation of a *send* action.

```
Process::
void write (UUID srvId, Dataset data) {
  Key puK_ = getServicePublicKey(srvId);
  EncDataset dataEnc = encrypt(puK_, data);
  new Launcher().write(srvId, this.progId, dataEnc);
}
Launcher::
void write (UUID srvId, UUID progId, EncData data) {
  Service srv = lockupService(srvId);
  SignedCertificate signedCert = getCertificate(progId);
  new Service().post(signedCert, data);
}
```

```
Service::
void post(SignedCertificate signedCert, EncData data) {
  boolean verified = verifyCertificate(signedCert);
  if (verified) {
    Dataset data = decrypt(this.prK_, data);
    storeLocalData(data);
  }
}
```

Listing 1. API operations (in bold) to implement the *send* action.

The *send* action begins with the execution of the `write()` operation in the `Process` class, taking as parameters the ID of the target service and the dataset to be sent to the service. This operation starts by retrieving the service's public key, the it encrypts the dataset, and creates a `Launcher` object to invoke the `write()` operation on it by passing as parameters the ID of the service, the ID of the process, and the encrypted dataset. The `write()` operation in the `Launcher` class starts by locating the target service and retrieving the integration process's attestation certificate. It then creates a `Service` object to invoke the `post()` operation, forwarding to the service the signed certificate and the encrypted dataset. The `post()` operation begins by validating the attestation certificate, then it decrypts the dataset, and so securely stores the data locally.

3. Conclusion and Future Work

We present *iDevS*, a domain-specific, technology-agnostic API to secure integration processes in TEEs, mitigating data exfiltration during execution. The iDevS API abstracts integration actions, such as *receive* and *send*, from the TEE execution and attestation mechanisms, enabling deployment across different TEEs without requiring modifications to the integration logic. We validate its effectiveness through a study of a case implemented with Morello Board compartments. Future work will assess the generalisability of the API in TEEs such as Intel SGX, AWS Nitro Enclaves, and ARM TrustZone.

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].