UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Herman Rull

Towards practical privacy-preserving data analysis with Intel TDX-based Sharemind HI

Master's Thesis (30 ECTS)

Supervisors: Armin Daniel Kisand, MSc
Raimundas Matulevičius, PhD

Tartu 2024

Towards practical privacy-preserving data analysis with Intel TDX-based Sharemind HI

Abstract:
Making evidence-based decisions often requires combining data of different data owners. Due to privacy concerns, data owners may be reluctant to share their data. Sharemind HI is a platform for developing data analysis applications that protect data owners' data throughout its lifecycle. Data-in-use protection is guaranteed by running operations involving data owners' data in Intel SGX TEE.

The problem with Sharemind HI is that developing analysis code on it was complicated due to programming language and library constraints of Intel SGX. Here we propose a Sharemind HI architecture built around Intel TDX TEE technology. The architecture demonstrates how the development limitations present in the old system could be eliminated without losing any core functionality. Additionally, we found that risks caused by Intel TDX's lack of isolation and data sealing can be managed using technical controls. Intel TDX-related overhead in expected Sharemind HI data flows is minimal compared to running them in a regular VM.

The resulting architecture is a step towards Sharemind HI that supports a wider range of programming languages and libraries, making the life of an analysis code developer easier as he can employ tools that best suit the situation. The architecture can be used to evaluate the system further or serve as a guiding document for implementing the new system.

Keywords:
trusted execution environments, software architecture, privacy-preserving technologies

CERCS: P170 Computer science, numerical analysis, systems, control.

Praktilise privaatsust säilitava andmeanalüüsi suunas Intel TDXil põhineva Sharemind HIga

Lühikokkuvõte:
Tõenduspõhiste otsuste tegemiseks on sageli vaja liita erinevatelt osapooltelt saadud andmeid. Samas ei pruugi andmeomanikud olla privaatsusest tulenevatel kaalutustel nõus oma andmeid jagama. Sharemind HI on platvorm, er arendada andmeanalüüsi rakendusi, mille eesmärk on säilitada andmeomanike andmeid andemete elukaare vältel. Seejuures kaitse parasjagu töödeldavatele andmetele on tagatud jooksutades arvutusi Intel SGX usaldatavas täitmiskeskkonnas.

Intel SGX piirab programmeerimiskeelte ja teekide kasutust. Piirang laieneb ka Sharemind HIga arendatud andmeanalüüsi rakendustele. Töös pakume välja tarkvaraarhitektuuri Sharemind HI-le, mis kasutab Intel TDXi usaldatavat täitmiskeskkonda. Loodud arhitektuur võimaldab eemaldada praegusele süsteemile rakenduvad arenduspiirangud, seejuures säilitades vana süsteemiga võrdväärse võimestiku. Me leidsime, et Intel TDXist puuduvatest isolatsiooni ja andmete pitserdamise võimekusest tingitud riske on võimalik maandada kasutades tehnilisi meetmeid. Jooksutades Sharemind HI andmevooge Intel TDXiga ja tavalises VMis, nägime, et Intel TDXi kasutamisega seonduvad lisakulud on minimaalsed.

Töös loodud arhitektuur on samm erinevaid programmeerimiskeeli ja teeke toetava Sharemind HI suunas. See lihtsustab andmeanalüüsi arendajate tööd, võimaldades kasutada olukorrale sobivaid tööriistu. Magistritöös kirjeldatud arhitektuuri on võimalik kasutada süsteemi edasiseks hindamiseks või uue süsteemi implementeerimise toetamiseks.

Võtmesõnad:
usaldatavad täitmiskeskkonnad, tarkvaraarhitektuur, privaatsust säilitavad tehnoloogiad.

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, kontroll.

Contents

Acronyms

AD Architecture Description

AEAD Authenticated Encryption with Associated Data

BPMN Business Process Modelling Notation

CAPEC Common Attack Pattern Enumerations and Classifications

CMAC Circular Message Authentication Code

DA Data Analysis

DEK Data Encryption Key

DFC Dataflow Configuration

FUSE Filesystem in userspace

HI Hardware Isolation

IS Information System

ISSRM Information System Security Risk Management

KBS Key Brokering Service

RA Remote Attestation

SDK Software Development Kit

SEAM Secure Arbitration Mode

SGX Secure Guard Extensions

TD Trusted Domain

TDX Trust Domain Extensions

TEE Trusted Execution Environment

UML Universal Modelling Language

VM Virtual Machine

VMM Virtual Machine Monitor

1 Introduction

Sharemind Hardware Isolation (HI) is a platform for developing data analysis applications handling sensitive data. Rather than relying on conventional trust-based mechanisms to protect end users' data, Sharemind HI enforces that the data is protected by encryption throughout its lifecycle [7]. Using Sharemind HI to protect end users' data can help businesses comply with regulations, creating unique selling propositions and convince privacy-conscious data owners to share their data. The security guarantees offered by Sharemind HI are possible thanks to using Intel Secure Guard Extensions (SGX) Trusted Execution Environment (TEE) technology for performing operations on confidential data [7]. The software development for SGX enclaves includes Intel maintained Software Development Kit (SDK), which compiles C or C++ code to machine code that their TEE could run. In addition to limiting the programming language choice, the SDK supports only libraries previously adapted by Intel [18]. These limitations extend to data analysis programs, making developing analysis code for Sharemind HI more complex.

In this thesis, we investigate using Intel Trust Domain Extensions (TDX) in Sharemind HI as the TEE provider to lower the complexity that Sharemind HI integration currently has. Intel TDX is a relatively new technology, released only in 2023 in its 4th Generation Scalable Xeon CPUs [16]. The runtime element of Intel TDX is Trusted Domain (TD), a Virtual Machine (VM) run inside TEE [16]. As such, TDs are very flexible, allowing programs written in any programming language, using any library, to run inside. However, transitioning from one TEE to another is not a trivial step, as the overall system needs to be adapted according to the properties of the new TEE to preserve functionality, performance, and security guarantees.

The main research question of the thesis is: "How could Sharemind HI use Intel TDX to simplify task development?" The question is further split into the following supporting research questions (SRQs):

- SRQ1 - What are the means of creating a verifiable model for TDX-based Sharemind HI? We capture requirements of the domain and peculiarities of Intel TDX that the eventual model must consider. We then introduce the methods for creating a model that can be used to verify the fulfilment of the requirements.
- SRQ2 - What is the architectural description of TDX-based Sharemind? We construct an architectural description using the method found in SRQ1. The answer to this supporting research question is the contribution of the thesis.
- SRQ3 - How does the new architecture solve the requirements? We verify the architecture found in SRQ2 against requirements complied in SRQ1 using methods described in SRQ1.

In Chapter 2, we give an overview of TEEs and Data Analysis (DA) domain in which the Sharemind HI operates. Furthermore, we explain the methods that were

used to construct and evaluate the architecture. Chapter 3 constructs the architecture of the TDX-based Sharemind HI. In Chapter 4, we verify the contribution against requirements. Chapter 5 concludes the work by answering the research question and critically evaluating contributions and potential improvements. A comprehensive view of the research questions and corresponding chapters is given in Table 1.

Table 1. Thesis structure given as a mapping between research questions chapters.

Identifier	Research question	Chapters
MRQ	How could Sharemind HI use Intel TDX to simplify task development?	
SRQ1	What are the means of creating a verifiable model for TDX-based Sharemind HI?	2. Modelling Software Systems
SRQ2	What is the architectural description of TDX-based Sharemind?	3. Constructing the Architecture
SRQ3	How does the new architecture solve the requirements?	4. Verifying the Architecture

2 Modelling Software Systems

This chapter is based on the first supporting research question - What are the means of creating a verifiable model for TDX-based Sharemind HI? The question can be answered by answering the following subquestions: (1) What are the key requirements that the model must capture? (2) How to model the system using software architecture? and (3) What are the means of verifying the architecture?

2.1 State of the Art

In the first part of the chapter, we give an overview of TEEs with a focus on Intel TDX and Intel SGX technology. In the second chapter, we explain the SGX-based Sharemind HI architecture. In the third chapter, we state the goals of the model.

2.1.1 Trusted Execution Environment

Functionally, TEEs enable running workflows in untrusted environments given that the user trust the implementation of TEE. More concretely however, Sabt et al. [32] explain the TEE properties as follows:

- the authenticity of the code executed in it;
- the integrity of its runtime states (e.g. CPU registers, memory and sensitive I/O);
- the confidentiality of information stored on persistent memory;
- Remote Attestation (RA) capability;
- resistance against all software attacks;
- resistance against physical attacks against the main memory of the system;
- resistance against attacks using backdoor security flaws.

RA enables extending the trustworthiness of a TEE and its workload to external third parties. Bartock et al. from the National Institute of Standards and Technology (NIST) define attestation in Trusted Cloud report [1] as taking measurements of the TEE and workload and digitally signing them. The recipient of the attestation Quote then verifies the signature and measurements. The common terminology is to refer to the party which starts RA to verify workload in the external device as Relying Party, and the party which provides a Quote as Attester [2].

While both Intel SGX and Intel TDX are TEEs, they differ in terms of their TEE. Intel SGX technology enables running enclaves, which are process-sized TEEs [5]. On the other hand, Intel TDX TEE is TD, which is a VM [4]. The size of TEE scopes the

workloads that can be run inside it. A large TEE is therefore more flexible, however it also makes TEE-level isolation of processes less performant. To isolate processes using TDs, each process would have to have its own OS.

Besides TEE size, we compare Intel TDX and Intel SGX technologies based on 5 building blocks identified by Sabt et al. in [32]. These building blocks are secure scheduling, secure boot, secure storage, inter-environment communication and trusted I/O path.

Secure scheduling concerns itself with how shared resources, notably memory and CPU states, are managed to avoid unauthorised access and tampering. In Intel TDX, the resource management of TDs is handled by untrusted Virtual Machine Monitor (VMM). All communication between VMM and TD is relayed through TDX Module. The TDX Module and TDs run in separate processor mode called Secure Arbitration Mode (SEAM). SEAM mode has a separate memory range, which is encrypted and integrity-protected using Intel MK-TME technology. Only other processes running in the SEAM mode can read and write to memory addresses in SEAM memory range. For each TD, the TDX Module lets VMM generate private memory to separate it from other TDs running in the SEAM mode. When a TD exits, its CPU states are stored in the trusted domain's private memory space and then wiped from the CPU.[4]. In Intel SGX, each enclave had its own protected memory, which is only accessible from within the enclave. When the computation flow exits the enclave, the CPU states are saved in the protected memory [5].

Secure boot guarantees that only verified code can be started. Since the verification is always done by some code, there must be some initial piece of logic which acts as a root of trust. The root of trust is considered to be *a priori* trustworthy [32]. Both Intel SGX and Intel TDX have ingrained the root of trust into the hardware [4, 5]. During RA, trusted hardware packs, among other data, measurements of the loaded software inside the Quote.

Secure storage enables storing data persistently across system restarts with the guarantee that only configured parties can decrypt it. This feature is not supported in Intel TDX out of the box. To achieve data persistence across restarts, Intel recommends using Key Brokering Service (KBS) [20] to provision a key early in the VM start-up process. The provisioned key can then be used to decrypt the encrypted disk partition. Intel SGX offers secure storage capability based on data sealing. Enclave can request the CPU to encrypt the data according to some policy, which determines which enclaves can decrypt it.

Inter-environment communication describes whether a TEE can communicate with other TEEs and regular runtime environments run on the same system. If a TD needs to communicate with other TDs or legacy VMs, the communication shall be done over regular network [21]. Intel SGX also provided a mechanism for enclaves to communicate with each other. In contrast to TDs, enclaves operate in the same OS and as such, regular

inter-process communication mechanisms could be used to transmit data between them.

Trusted I/O path is an authenticated and confidential path between I/O devices and TEE. Intel TDX does not support Trusted I/O path, though Intel has roadmapped TEE I/O for its TDX 2.0 release [19]. Until then, any I/O device communication is done using shared memory. The integrity, confidentiality and authenticity of all messages sent over I/O must be implemented through custom means. Intel SGX supports some trusted paths with its closed-source Protected Audio Video Path (PAVP) module [15]. However, Intel SGX does not general trusted I/O paths.

Table 2. Comparison between Intel SGX and TDX extensions.

Category	Intel SGX	Intel TDX
TEE size	Process	Virtual machine
Secure Boot	HW-based root of trust	HW-based root of trust
Secure Storage	Supported	Not supported
Secure Scheduling	Memory and CPU states	Memory and CPU states
Inter-environment communication	Supported	Supported
Trusted I/O Path	Limited use cases	In TDX 2.0

2.1.2 Sharemind Hardware Isolation

Note: in this chapter Sharemind HI refers to SGX-based Sharemind HI, in the rest of the thesis, Sharemind HI refers to TDX-based Sharemind HI.

Sharemind HI [6] is a platform for developing privacy-preserving data analytics applications. The data analytics application has a central server, which stores all uploaded data and allows application specific Tasks to perform analytics on this data. Only select users of the application can consume the output data. The privacy-preserving guarantees are achieved using Intel SGX, access controls (called Dataflow Configuration (DFC)) and organisational controls.

There are two types of stakeholders in Sharemind HI-powered solutions - organisational and application stakeholders (see Table 3). Organisational stakeholders are concerned with developing the data analysis application. The organisational stakeholders are Sharemind HI developer, Task developer and Coordinator. Sharemind HI developer main responsibility are maintaining Sharemind HI Client and Server. Coordinator is responsible for developing the project-specific application. In DFC, he defines the Tasks and End-users of the applications and assigns them roles and permissions. Task developers are responsible for implementing Tasks as specified by the Coordinator. Application stakeholders are also called End-users, as they are the users of the application created by organisational stakeholders. There are 6 End-user roles: Data Producer, Data Consumer, Task Runner, Enforcer, Administrator and Auditor. Single End-user can have multiple different roles. Data Producer can upload data to data analysis Tasks, Task Runner can

start Tasks, Data Consumer can download data, Auditor can inspect the application runtime logs, Enforcer can verify DFC-s and Administrator can monitor application, create backups, start recovering from backups and start DFC upgrades.

Table 3. List of stakeholders and their responsibilities in Sharemind HI powered application. An application end-user can have multiple roles. For example, Data Producer and Task Runner are commonly assigned to the same End-user.

Roles	Responsibilites and Concerns
Organizational stakeholder	
Coordinator	Finding business case, defining end-users, their roles and permissions in DFC
Sharemind HI developer	Developing new features, finding and fixing bugs
Task developer	Developing and deploying task
Application stakeholder	
Data Producer	Uploading data
Data Consumer	Downloading output
Task Runner	Starting tasks in TEE
Auditor	Inspecting runtime logs
Enforcer	Verifying DFC
Administrator	Monitoring application, creating backups, recovering from backups

The access controls of Sharemind HI-powered application are described in a single configuration file called DFC. DFC defines the Tasks and End-users of the system. Additionally, it contains the permissions of end-users and Tasks at the topic level. Topics are lists that contain data entries of a certain type. In Fig. 1, Stakeholder1 is Data Consumer in DataTopicA and has permission to run TaskX. TaskX is a Data Producer for DataTopicA. Users of the system can verify the enforcement of DFC and authenticity of Tasks using RA.

Before a Server with a certain DFC is deployed, the DFC must be approved by all enforcers [6]. Enforcers must verify that the Tasks are valid. To do that, they build the Tasks locally and see if their fingerprints match the ones given in DFC. Additionally Enforcers verify that End-user real certificates match the certificate given in the DFC. This is done by receiving Public Keys from Stakeholders over secure side channels.

Analysis code in Sharemind HI is run in Tasks. Tasks have a single run method that can be invoked by Task Runners. To access Data Producers' encrypted data from persistent storage, they can use special system methods that handle encryption and decryption for them. Implementation wise, Tasks are enclaves running in separate processes. This means that the isolation from each other and the rest of the Sharemind HI is implemented at the TEE level.

2.1.3 Purpose of the Model

The model aims to prove that Intel TDX can be used in Sharemind HI to ease the development of Tasks. More concretely, the model should be capable of evaluation

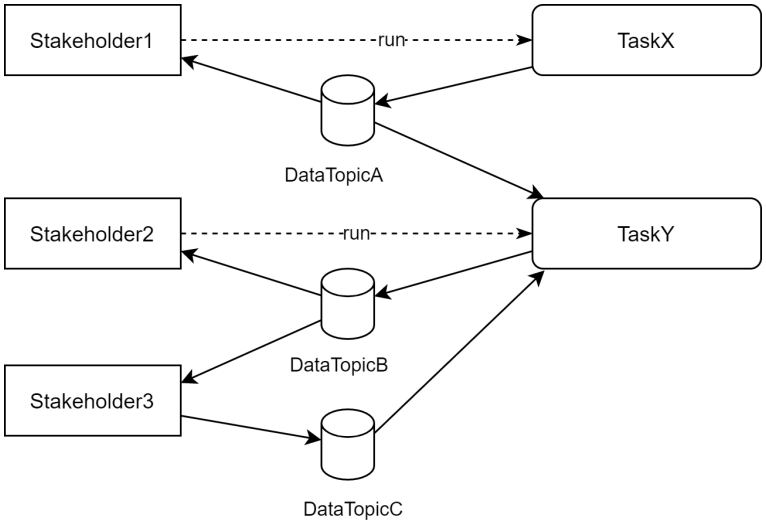

Figure 1. DFC visualized as a graph [6]. The pointed line to the data topic indicates being a Data Producer for this topic. A pointed from data topic means that the targeted Task or Stakeholder is a Data Consumer in this topic. The dotted line to Task shows permission to run this particular Task.

following five aspects:

- Feature Parity - does the TDX-based Sharemind HI Server have the same functionalities as its SGX-based counterpart?
- Secure Storage - Intel TDX does not support secure storage. Can data be persistently stored without secure storage, what are the risks and how are they mitigated?
- Isolation of Tasks - Intel TDX operates at VM level. How can the enclave-based isolation zones of SGX-based Sharemind HI be replicated using Intel TDX
- Performance Overhead - what is the overhead of using TDX-based Sharemind HI?
- Simplifying Task Development - how does TDX-based Sharemind HI remove programming language constraints and how does it simplify Task development?

2.2 Architecture as a Model of the System

ISO/IEC/IEEE 42010 [23] standard defines software architecture as "fundamental concepts or properties of an entity in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes". The purpose of the software architecture to give a holistic yet simplified view of the system. In the thesis, we aim to create an architecture that shows that TDX-based Sharemind is capable of solving key requirements captured in the previous chapter.

2.2.1 Architecture Description

In [31], Rozanski and Woods propose a framework for describing architecture. The resulting document, Architecture Description (AD), is composed of different products, two of which - viewpoints and scenarios - will be used in this thesis. The framework was chosen because it explains how the products can be used to verify the architecture. Furthermore, it had clear guidelines and checklists to support the process of creating AD.

Viewpoints Software systems can be complex. Expecting one architectural model to address the concerns of all stakeholders simultaneously is unrealistic. Kruchten [26] proposes a solution to this problem by splitting the architecture into different viewpoints in his "4+1" model [26], each viewpoint having a particular part of the system it captures. [31] go a step further and propose slicing the architecture into seven orthogonal viewpoints as follows:

- Context viewpoint describes the external entities with which the system interacts.
- Functional viewpoint describes core functional elements of the systems as well as the interfaces between them.
- Information viewpoint describes how the system stores, manipulates, manages and distributes information.
- Concurrency viewpoint maps functional elements to concurrency units and describes which units can execute concurrently.
- Operational viewpoint describes how the system will be managed during its runtime.
- Deployment viewpoint captures the environment in which the system will be deployed.
- Development viewpoint describes how the described architecture should be split between modules.

Scenarios An architectural scenario or simply scenario is a description of an interaction between the system and an external entity. It captures a situation in which the system will likely find itself and how the system should respond to the situation [31]. The scenarios are divided into functional scenarios and system qualities scenarios based on their focus.

Functional scenarios focus on explaining what the expected response to external stimuli should be. Most of the time, they are derived from the use cases [31] and cover the following aspects:

1. Overview of the situation that the scenario illustrates;

2. System state at the start of the scenario with emphasis on stored information;
3. Any significant changes in the system environment at the start of the scenario;
4. External stimulus, which caused the scenario to occur;
5. Required system response from an external observer perspective.

System quality scenarios focus on how the system should respond to changes in the environment with the emphasis lying on the effect this change has on the quality attributes. A system quality scenario should explain:

1. Overview of the situation that the scenario covers;
2. System state at the start of the scenario with emphasis on system-wide state;
3. Any significant changes in the system environment at the start of the scenario;
4. Description of the change in the environment;
5. Required system behaviour, given a change in its environment.

2.2.2 Modelling Languages

Viewpoints and scenarios describe what objects should be modelled. Purposefully, they leave the question of how to model these objects up to the implementer [31]. In this thesis, we use both non-formal models and formal methods to describe the objects. The non-formal models are described as they come up. The formal models are based on Universal Modelling Language (UML) and Business Process Modelling Notation (BPMN). Both UML and BPMN are defined and managed by Object Management Group [29], and more detailed information about these methods can be found in their documentation. UML and BPMN were chosen due to their thorough and example-rich documentation. The BPMN diagrams were constructed using Camunda [3]. The rest of the models were drawn using drawio [24].

2.3 Architecture Evaluation Methods

Architecture is an abstract model of the real system. It focuses on aspects considered relevant and leaves details up to the implementation. As such, it risks making irrelevant or even wrong abstractions. The purpose of architecture validation methods is to make sure that the abstraction are reasonable and technically correct [31], thereby increasing the confidentiality that the architecture is representative of the actual system.

There exist various different methods for validating software architecture. They vary in their goals, approaches and level of depth. Depending on the project and stage of the

development Rozanski and Woods recommend using different methods. For small-scale high-risk systems such as TDX-based Sharemind HI, they recommend using scenario-based evaluation, prototypes and skeleton systems [31]. The first two methods should be prevalent when creating an architecture description. Skeleton systems should be the go-to method as the project transitions from the architecture development phase to the system implementation phase. In the thesis, we follow their recommendation to the extent of using scenarios and prototypes to evaluate the system. Since the actual system development is beyond the scope of the thesis, the skeleton system validation is left as future work.

2.3.1 Scenario-based evaluation

Scenario-based evaluation is based around scenarios described in Section 2.2.1. After identifying the critical scenarios, the architecture is evaluated on the basis of how well it is capable of tending to the situations described in the scenarios. Scenario-based evaluation is relatively easy to apply and, at the same time, gives valuable information about the effects and trade-offs of architectural decisions [31]. However, scenarios are relatively high-level models leading to non-precise results.

2.3.2 Prototyping

Prototyping is an evaluation method where select parts of the architecture are implemented in a throw-away fashion, e.g. after the evaluation is done, the prototype is discarded. Compared to scenario-based evaluation, prototypes mock the real system more closely, making results obtained through prototyping more convincing. On the other hand, prototyping requires more effort, leading to higher costs and longer development times.

2.3.3 ISSRM Domain Model

With trusted persistent storage and isolation of Tasks being pushed to Sharemind HI level, there will be additional security risks the system must mitigate. To describe the risks that these changes cause, we employ Information System Security Risk Management (ISSRM) domain model. The domain model identifies and defines common concepts used in existing standards and security risk management methods [8]. The concepts and their definitions are as follows:

- Threat agent - an agent that triggers the threat to cause harm to Information System (IS).
- Attack method - description of how the threat agent carries out the threat.
- Vulnerability - Weakness in the IS.

- Impact - negative consequence following the accomplished threat.
- IS asset - a component or part of IS affected in the risk.
- Business asset - informational element having value due to business model of the IS.
- Risk treatment - how the identified risk is being handled. Possibilities are avoiding the risk, reducing the risk, transferring the risk and accepting the risk.

The meta-model was chosen due to its good performance in a systematic overview [12] by Ganji et al. and due to prior experience of the supervisors.

2.4 Chapter summary

This chapter investigated how to create a model to verify the possibility of using Intel TDX in Sharemind HI. In the first subchapter, we explained what Intel SGX, Intel TDX and Sharemind HI are and outlined 5 key requirements the model has to address. In the second subchapter, we introduced a viewpoint-based software architecture framework and modelling languages that will be used for constructing the model. In the third subchapter, we described two methods for verifying the model.

3 Constructing the Architecture

This chapter deals with the second research question - What is the architectural description of TDX-based Sharemind? The architecture of the system is explained using 4 different views that conform correspondingly to Context, Functional, Information and Concurrency viewpoints. Operational, Deployment and Development viewpoints were left out of the scope as they refer to implementation-specific details. The subquestions that the chapter answers are as follows: (1) How does TDX-based Sharemind HI look like from a Context viewpoint? (2) How does TDX-based Sharemind HI look like from Functional viewpoint? (3) How does TDX-based Sharemind HI look like from Information viewpoint? (4) How does TDX-based Sharemind HI look like from Concurrency viewpoint?

3.1 Context View

The purpose of the context view is to model Sharemind HI as a black box and capture external entities with (1) whom the system interacts, (2) what are the messages and (3) the required properties of the channel through which the messages are sent.

In total, there are 6 different external entities/environments that interact with either TDX-based Sharemind HI Server or Client-Key Brokering System, Attestation Provisioning System, Persistent Storage, End-user and System administrator. In the rest of the subchapter, they are described more thoroughly. The Fig. 2 summarizes the whole subchapter.

Key Brokering Service A system responsible for storing and retrieving the persistent keys of TD. This component is necessary as Intel TDX does not provide persistent storage, and key provisioning by an Administrator would mean that the Administrator would have unrestricted access to persistent data. The latter goes against the purpose of Sharemind HI. There are two types of requests that move between KBS and Sharemind HI server - key store and key retrieve requests. Both requests are initialised at the Sharemind HI server, and KBS provides responses. When Sharemind HI Server is first started, it stores a random master key in the KBS. After a reboot, the Sharemind HI Server requests the key from the KBS. The secrecy of the master key is vital as it acts as a root of trust. The communication with KBS must be mutually attested, encrypted and tamper-proof. Through attestation, Sharemind HI Server verifies the trustworthiness of KBS. On the other hand, the KBS uses Sharemind HI Server attestation result (Quote) as an authentication method - should the Quote change, the keys stored will be inaccessible. As a result, after DFC upgrades, all user data uploaded to Persistence Storage is lost by default.

The KBS ecosystem is relatively new, and as such, there are not many commercially available off-the-shelf solutions. One notable solution, Trustee, developed by Confidential Containers [**noauthor_confidential-containerstrustee_2024**] provides tools

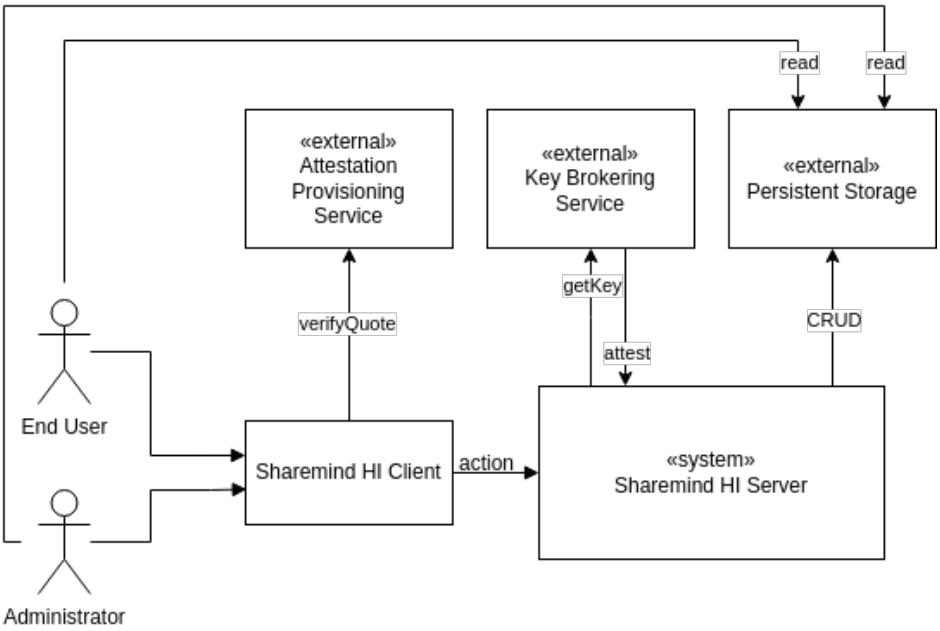

Figure 2. Context view of the Sharemind HI capturing external entities and environments, and interactions between them

for creating a KBS capable of returning keys against a Quote. Running Trustee in a TEE, opens up the possibility for Sharemind HI Server to also verify the Quote of Trustee-powered application, verifying that the latter does not leak keys in any way.

Attestation Provisioning System. Attestation Provisioning System is responsible for verifying RA Quotes. It is necessary as it confirms to end users that a certain Quote was signed by a certain SGX-powered chip. The Attestation Provisiong System is built upon Intel Data Center Attestation Primitives. The interactions with the Attestation Provisioning System are done over the network, and the messages must be tamper-proof and authenticated. The nature of the communication is based on Sharemind HI Client/Server requests and system responses.

Persistent Storage. When KBS is required to store secrets across restarts, the Persistent Storage stores the actual data. Two broad categories of requests are associated with Persistent Storage - storing an retrieving data. In both cases, the Persistent Storage acts as the responding party. The communication channel between Persistent Storage and its users does not have any requirements, as Persistent Storage is already considered to be untrusted. Instead, to achieve integrity and confidentiality of data in Persistent Storage, we employ encryption and MACs.

End-user. End users are the eventual users of DA application. They have a version of Sharemind HI Client that they can use to make requests. In contrast to previous entities, which responded to Sharemind HI Server requests, the client always initialises

communication between End-user and Server. The communication channel must be tamper-proof, encrypted, client-authenticated and the server must attest itself to the client.

System administrator. System administrator supports the DA application during its runtime. They are responsible for handling upgrades of DFC and monitoring the system. The communication channel must be tamper-proof, client-authenticated and the server must attest itself to the client.

3.2 Functional View

. The purpose of the functional viewpoint is to pick the black box seen in the context viewpoint apart into runtime parts called functional elements. Functional elements handle concrete functionalities that map to software code modules [31]. Functional elements expose interfaces through which they can be accessed by other elements.

The set of functional elements identified for TDX-based Sharemind HI and their responsibilities is given in the list below:

- **Gateway** manages communication with the client application. It serves as an entry point to the Sharemind HI Server - it authenticates the user and handles calling requested functionalities;
- **Attestation Module** handles RA Quote generation;
- **Persistence Controller** is responsible for saving and retrieving data from persistent storage. It also handles the retrieval and creation of the Master Key.
- **DFC Enforcer** keeps track of the DFC history and handles DFC upgrades. Based on the most recent DFC authenticates and authorises users;
- **Task Manager** manages Tasks. This includes running and sending messages to Tasks, handling Task setup and failures;
- **Task** is DA application-specific analysis code. It performs operations on uploaded data and/or additional parameters passed to it with a Task run call.

Functional elements communicate with each other over interfaces. In total, we identified 9 necessary interfaces that the functional elements must expose. 6 interfaces are completely internal. They enable communication between functional elements. The remaining 3 interfaces are between an internal functional element and an external entity. Internal interfaces mostly connect elements running in the same process and, as such, can be called using function calls. Non-internal interfaces are mostly implemented over HTTP. Fig. 3 displays the functional elements and their interfaces with each other. The rest of the chapter defines each interface at function intent and signature level.

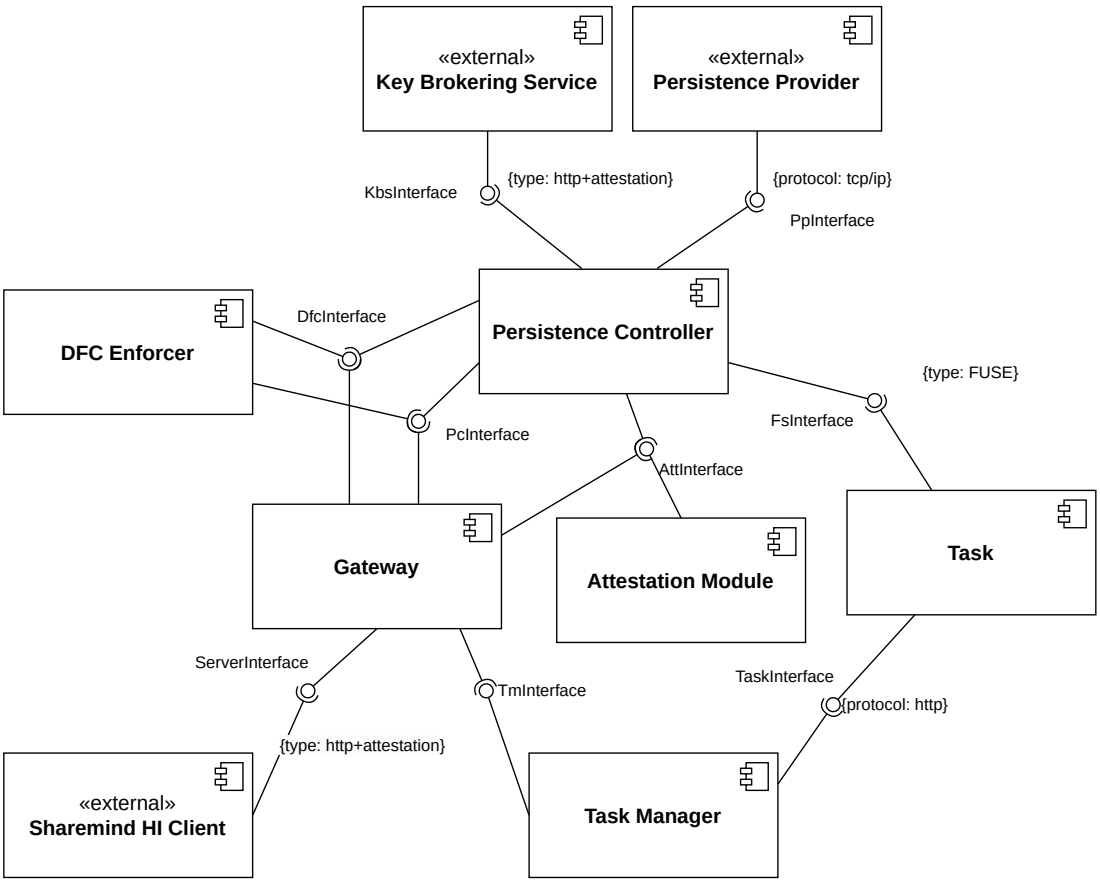

Figure 3. UML component diagram displaying functional elements and their interfaces.

The functional elements and their interfaces are displayed in the Fig. 3.

ServerInterface. The interface is exposed by Gateway for Administrators and End-users, who, using Sharemind HI Client library, send HTTP requests to the Sharemind HI Server. Sharemind HI Client is set up such that it requires attestation proof from Sharemind HI Server before allowing the use of other functionalities. The interface is composed of following ten functions:

```
(Status, ServerSessionPublicKey) createSession1(challenge: Challenge, pub_key:Public
 (Status, Quote) createSession2(sess_pub_key:SessPublicKey)
 (Status, DEK) uploadData(confidential_data: byte[], topic: uuid)
 (Status, DataEntry) downloadData(data_entry_id: uuid)
 (Status, Optional<byte[]) runTask(
     task_id: uuid,
     additional_args:Optional<byte[]>
```

```
)
Status uploadDfc(dfc: Dfc)
Status getStagingDfc()
Status verifyDfc(verification_result:boolean)
Status upgradeDfc(dfc:Dfc)
Status startRecovery()
```

createSession is a two-round-trip function through which the Client and Server application establish a symmetric key. It is based on Sign-and-MAC (SIGMA) family of key exchange protocols proposed by Krawczyk [25]. During the second round-trip, Gateway extends the key exchange response with an attestation Quote as described in Intel SGX attestation documentation [17]. The symmetric key serves as a basis for sending authenticated encrypted and tamper-proof messages between server and client. *uploadData* function is used to upload data. In response, the user gets Data Encryption Key (DEK) that was used to decrypt the data in the persistent storage. The key is deterministically generated based on a nonce and Master Key. This key can be used to regrant the access to Sharemind HI Server DFC is upgraded and it loses its Master Key. *downloadData* function is for retrieving data entries from Persistent Storage. *runTask* can be used to run Tasks. Besides having to specify the task, Task Runner can pass in additional variables. The response to the function depends on the Task. For long running tasks it could be a verification of starting the computation, for short Tasks it can contain result of the analysis. *uploadDfc* is used to upload new DFCs, which can then be queried using *getStagingDfc* and verified using *verifyDfc*. If all Enforcers have approved, the DFC Administrator can trigger the upgrade using *upgradeDFC*. *startRecovery* is a function for restarting the Server after a failure. Status is an enum describing the result of the request in this interface as well as the following interfaces.

AttInterface. The interface is composed of two functions:

```
boolean verifyQuote(Quote: Quote)
Quote getQuote(target: EndUser|KBS )
```

verifyQuote function is used to verify Quotes provided to Sharemind HI Server. One example of such Quote provider is KBS. *getQuote* is used during RA to construct the Quote. Depending if the target is End-user or KBS, some fields of the Quote vary. *generateKeyPair* is used during RA for creating a temporary key pair for establishing a Session Key. The interface is used by the Persistence Controller and Gateway. **TmInterface**. The interface is composed of one function:

```
(Status, Optional<byte[]) runTask(
    task_id: uuid,
```

```
        additional_args:Optional<byte[]>
    )
```

runTask function of ServerInterface is forwarded to TaskMananger for implementation. Task Manager sends the arguments to the intended Task. The response from the Task is then relayed back to the Gateway.

DfcInterface. The interface contains four functions:

```
boolean isUserAuthorised(user: uuid, topic: uuid, operation: Upload|Run|Download
 Status updateStagingDfc(dfc: Dfc)
 Status fetchDfcHistory()
 Status addStagingDfc(dfc: Dfc)
```

isUserAuthorised function takes in three arguments and verifies if a user with uuid is allowed to perform an operation in a certain topic. *updateStagingDfc* starts pulling DFC-s from Persistent Storage. *fetchDfcHistory* takes a new version of the staging DFC and overwrites the existing copy. This action is supposed to be transactional in the sense that updating the local staging DFC and the version in the Persistent Storage are done in one transaction. *addStagingDfc* verifies the correctness of the DFC, uploads to Persistent Storage and then updates the local DFC History.

FsInterface. The interface is composed of the following three functions:

```
uuid[] listFiles(topic_name: string)
DataEntry retrieveData(data_entry_id: uuid)
Status storeData(data: byte[], topic: uuid, data_owner: uuid)
```

The functions are called when Task performs file system operations. *listFiles* is called when Task queries the contents of the file system. The response of the function is a collection of data entry uuids available to the Task. *retrieveData* is called when Task reads from a file. The function decrypts the data entry contents and returns it in decrypted form to the Task. *storeDataEntry* is called when Task writes results to the filesystem. The cleartext results are encrypted and persisted as data entry. The redirection of file system calls to Persistence Controller is done using Filesystem in userspace (FUSE). More information about FUSE is given in Appendix A. The authorisation of Tasks action is handled using DFC Enforcers *isUserAuthorized* function.

PcInterface The interface is composed of the following six functions:

```
DataEntry retrieveData(data_entry_id: uuid)
Status storeData(data: byte[], topic: uuid, data_owner: uuid)
Status deleteDataEntry(data_entry_id: uuid)
```

```
DfcHistory retrieveDfcHistory()
Status addStagingDfc(staging_dfc:Dfc)
Status updateStagingDfc(staging_dfc: Dfc)
MasterKey getMasterKey(dfc: Dfc)
```

The functions are called to perform create, read, update and delete operations on persistent data. The consumers of the interface are Gateway and DFC Enforcer. The Persistence Controller, the functional element that implements these functions, handles the encryption and decryption of the data to be operated on. Additionally, it handles connecting to different Persistence Storage Providers. The first 6 functions calls will be forwarded to Persistent Storage, however the last function, *getMasterKey* is to be performed against KBS.

TaskInterface. The interface is composed of one function:

```
byte[] run(additional_args:Optional<byte[]>)
```

The function is implemented in the Task by Task developers. The function is called by Task Manager using an HTTP request. The endpoint for the request is defined using a port assigned to the Task.

PpInterface and KbsInterface are implemented at the vendor side and are left out of the architecture description.

3.3 Information View

The purpose of the information viewpoint is to explain data entities of the system and give insights into their lifetime, ownership and identification.

Keys. There are numerous different keys used in TDX-based Sharemind HI with different lifetimes, so we look at two different categories of keys. One category of keys is related to persistently storing data, and the second category is related to session creation.

The Persistence key chain begins with the Master Key. The Master Key is created at KBS based on Sharemind HI Server Quote. It is then ephemerally stored in the Sharemind HI Server until the Server dies. Once the Server is restarted with a new DFC version, the Master Key is lost. Per application there exists one Master Key at a time. Each time data is uploaded, a new DEK is created based on the Master Key and a secret that will always be used to decrypt and encrypt the uploaded data. The DEK is encrypted using Master Key and the resulting eDEK is stored in Persistent Storage. The DEK is then returned to Client and wiped from the Server. DEK is introduced to enable quick restoring (only DEK needs to be sent to the Server) of Data Producers' Data Entries after DFC Upgrade, should the Data Producer wish that.

Session creation-related keys are used to authenticate the End user and the Sharemind HI Server. Each End-user of Sharemind HI has an asymmetric Client key pair that is used to authenticate them across sessions. This key pair is created during the deployment of Sharemind HI and remains constant across DFC upgrades and Server restarts. The public key of the key pair is fixed in DFC. On the other hand, Sharemind HI Server is authenticated by its hardware key and Intel Attestation Provisioning Service. The hardware key is ingrained in CPU fuses and can not be pulled or modified. Keys described so far are only used for authentication of users communication between Client and Server is protected by a symmetric Session Key. The Client and Server use temporary Session Creation Key Pairs to establish an ephemeral Session Key.

Table 4. Table showing persistence and attestation keys used in TDX-based Sharemind HI

Key	**Creation**	**Creation place**	**Storage**	**Lifetime**	**Count**
Persistence keys					
Master Key	Deterministically based on Server Quote	KBS	Ephemeral	DFC version	1
DEK	Randomly based on Master Key	Sharemind HI Server	Ephemeral	DFC version (extendable by Client)	1 per Data Entry
eDEK	Deterministically based on DEK and Master Key	Sharemind HI Server	Persistent Storage	DFC version	1 per DEK
Session creation keys					
Hardware key	Engrained in CPU during manufactury	Out of AD scope	CPU fuses	CPU lifetime	1 per CPU
Client key pair	DFC generation	Out of AD scope	End user local storage	Application lifetime	1 per Client
Client Session Creation Key Pair	Randomly during Session creation	Sharemind HI Client	Ephemeral	Session	1 per session
Server Session Creation Key Pair	Randomly during Session creation	Sharemind HI Client	Ephemeral	Session	1 per Enduser
Session Key	Session Creation based on exchanged Creation Keys	Sharemind HI Client and Server	Ephemeral	Session	1 per session per End user

Persistence.TDX-based Sharemind HI requires persistent storage of following three different types of data - DFCs, Data Entries, and the Master Key. Data Entry is a complex data structure containing encrypted data, data identifier, eDEK, topic identifier, retention date, data owner identifier and a MAC calculated over all other fields using Master Key.

DFC is a complex data structure containing 4 further each other referencing complex data structures - Stakeholders, Topics and Tasks and Approvals. Additionally each DFC holds a boolean whether a DFC is in staging phase an incrementing version number

a MAC that contains all other fields and Admin's signature of the MAC. The MAC is added by the uploading Admin.

The Master Key storage is handled by Key Brokering Service.

Attestation Structures. The result of the remote attestation is a Quote. Quote is a Report that has been signed by the hardware-specific key. The Quote can be verified in Attestation Provisioning System. If the Quote is correct, then the Report indeed describes the workflow of the attesting system. Report contains measurements of different parts of the system that can be compared by the Relying Party against some known good values. Additionally, Report contains a field called Reportdata, which, in comparison to other fields, can be constructed by the workload itself.

3.4 Concurrency View

Sharemind HI server is composed of one central Hub Process and at least one guest Task Process. Task Processes have two separate communication channels with the Hub Process, but no immediate methods to communicate with each other.

The Hub Process contains all TDX-based Sharemind HI logic. It hosts functional components Persistence Controller, Task Manager, Attestation Module, DFC Enforcer and Gateway. Depending on the capabilities of the underlying hardware, each functional component can employ multiple threads.

Task processes hold the Tasks and additionally must host an HTTP server. They are meant to listen on an HTTP port and, when a request arrives, then perform the analytics task. Depending on the request, the task process might want to fetch data from the database. Should a task die, then the server is responsible for logging the error and restarting the task.

The concurrency viewpoint is modelled in the following graph Fig. 4.

3.5 Chapter Summary

In the chapter, TDX-based Sharemind HI architecture was proposed. We modelled the system from different viewpoints, each explaining a different slice of the system. Context view gave a big-picture view of the external components which interact with the Sharemind HI Server. The Functional view listed 6 functional elements and 9 interfaces, which they expose. The Informational view explained different data objects used in TDX-based Sharemind HI. This included a description of different keys, persistently stored Data Entries and DFCs, and data structures used in RA flows. The concurrency view explained that Sharemind HI Server is composed of 2 two types of processes. There is one Hub process running the majority of functional components and multiple Task processes, each running a separate Task.

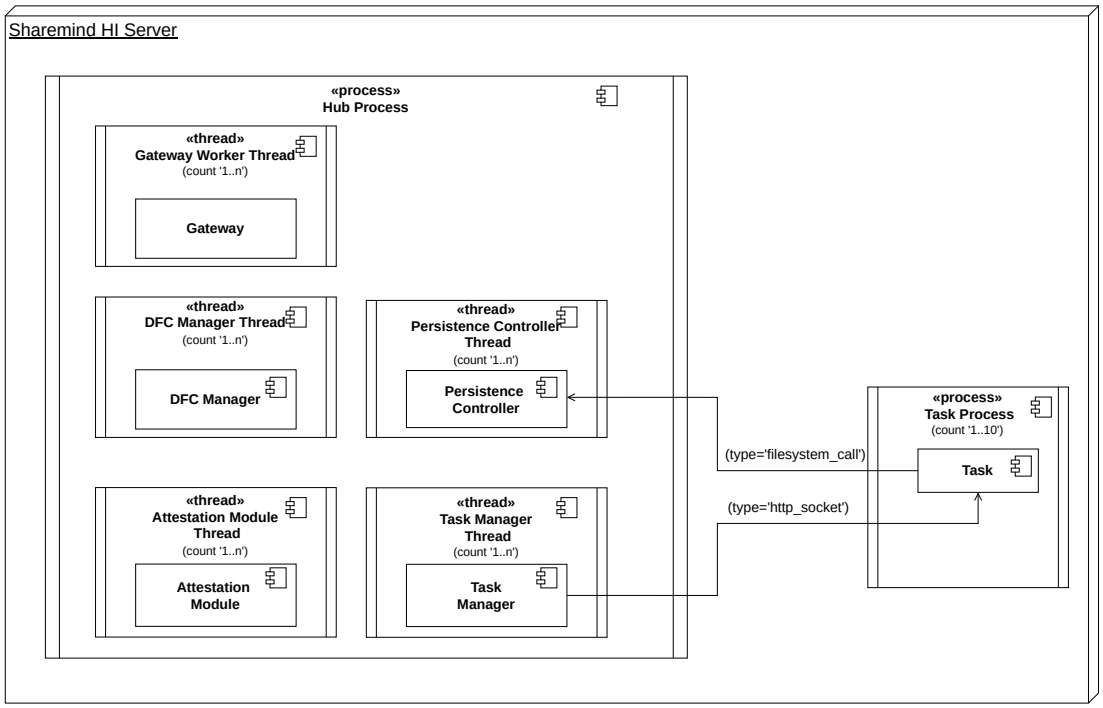

Figure 4. Model displaying functional elements of an architecture built around Intel TDX trusted domains

4 Verifying the Architecture

This chapter answers the last supporting research question - How does the new architecture solve the requirements?

It does this by answering the following subquestions:(1) How will the requirements be verified? (2) What are the results of scenario-based evaluation? (3) What are the results of prototype-based performance evaluation?

4.1 Creating Verification Plan for Requirements

The purpose of the architecture developed in the thesis is to show that Intel TDX could be used in Sharemind HI. More concretely, in Section 2.1.3, we defined 5 requirements that the architecture must be able to verify - Feature Parity, Unrestricted Task Development, Secure Storage, Isolation of Tasks and Low Performance Overhead.

To verify Feature Parity, we select core use cases of SGX-based Sharemind HI and construct functional scenarios based on them. We then create BPMN diagrams using context view elements as BPMN Pools, functional elements as BPMN lanes, functions as flow symbols between swimlanes and data entities as data objects. In the following list are the use cases and reasoning for their inclusion:

- Session creation - necessary prerequisite for other use cases
- Data download - necessary for data analysis purposes.
- Data upload - necessary for data analysis purposes.
- Run task - necessary for data analysis purposes.
- Verify DFC - necessary for explaining trust generation in TEE-based data analysis application.
- Start recovery - explains recovery of state across restarts in Intel TDX.
- Upgrade DFC - a complex process which requires verification.

For verification of Secure Storage and Isolation of Tasks, we formulate them as security quality scenarios. We find the parts of BPMN diagrams that are affected by the TEE change, find applicable attacks to these parts and formulate risks according to ISSRM domain model. Furthermore, we propose technical controls that could be used to mitigate these risks. Unrestricted Task Development requirement is verified by showing that the architecture nor the TEE restrict using programming languages commonplace in data science. To verify performance-related requirements, we create a prototype of the system according to the architecture. We then use the prototype to run end-to-end flows in both regular VM and TD. From that, we obtain Intel TDX overhead, which we can compare with that of Intel SGX

4.2 Verifying Scenarios

In this chapter, we show how the architecture described in Section 3 can be used to verify fulfilling functional, security and ease of development scenarios.

4.2.1 Functional Scenarios

The functional scenarios explain how the system should respond to given stimuli. In this chapter, we construct 7 scenarios based on the core use cases of Sharemind HI. We then verify using BPMN diagrams and architecture description constructed in Section 3 that the expected path from stimuli to response can be created. This verifies the requirement for feature parity because it shows that the new architecture can handle Sharemind core HI use cases.

Session creation. Client-server session creation in TDX-based Sharemind HI is a mutually authenticated key exchange composed of two round trips between the Client and the Server. During the second round trip, Server responds with a generated Quote. The main purpose of the Quote is to give proof to the client of the specific workload run on the server. Having a session is a prerequisite for all other client-server communication (except for querying DFC). It is invoked whenever a client makes a request requiring client-side authentication and no existing session exists. In the main success scenario, a symmetric key is created between Client and Server, and Client has verified Server's workload. A description of the scenario is given in Table 5.

Table 5. Scenario description of session creation given in a tabular format

Overview	Sharemind HI Client and Sharemind HI Server establish a trusted connection.	
System state	Sharemind HI Server application is running. No sessions exist between Server and participating Client.	
System environment	As usual.	
External stimulus	Client initialises the connection with Sharemind HI Server.	
Required system response	Main success scenario	Symmetric key is exchanged, client has established trust with Sharemind HI Server through TA.
	Suspicious Quote	Cancel session creation
	Server signature is revoked	Cancel session creation
	Server timeout	Cancel session creation.

The key steps of how the session creation process is handled in the architecture is visualised in Section 4.2.1 and given below:

1. Client initiates the client session creation handshake by sending the Challenge to the Server. This corresponds to *createSession* function.
2. Gateway receives the request and forwards it to Attestation Module for handling using *generateKeyPair*.
3. Attestation Module creates Session Creation Key Pair.
4. Attestation Module signs Server's session creation public key and a handshake-specific signature signed by hardware encoded secrets.
5. Gateway sends the signed public key to Client.
6. Client verifies the signature by the Attestation Provisioning Service. On failure session creation is stopped, on success the process continues.
7. Client creates Session Creation Key Pair.
8. Client derives the Session Key using the Server's Session creation public key and its own Session Key.
9. Client sends its session creation public key to Server.
10. Gateway derives Session Key and creates random bytestring as an identifier for the session.
11. Gateway requests Quote creation from Attestation Module using getQuote function.
12. Attestation Module session identifier starts Report creation and adds session identifier to the Report creation request. The Report is generated by the TDX Module.
13. Attestation Module uses TDX built-ins to turn the Report into a Quote. The Quote is signed using hardware secrets exposed through Intel TDX in-built functions.
14. Attestation Module returns the Quote to the Client
15. Client verifies Quote.

Data upload describes the process through which Data Producers their data to the Sharemind HI Server in order to make it available to analytics tasks. The process starts when Data Producer has entered data and topic to the Sharemind HI Client. In the main success scenario, the data is stored in Persistent Storage as a Data entry, and the Client receives DEK, which is used to encrypt the Data Producer's data. If Data Producer's session can not be found at the Server side, a new session must first be established. If Data Producer does not have permission to upload data to given topic, the process should be cancelled. Any timeout at the Client or Server side must trigger, ending the process. See Table 6 for a summarised scenario description.

Using the constructed architecture, the scenario can be completed as follows (see):

1. Client uploads confidential data using *uploadData* function.

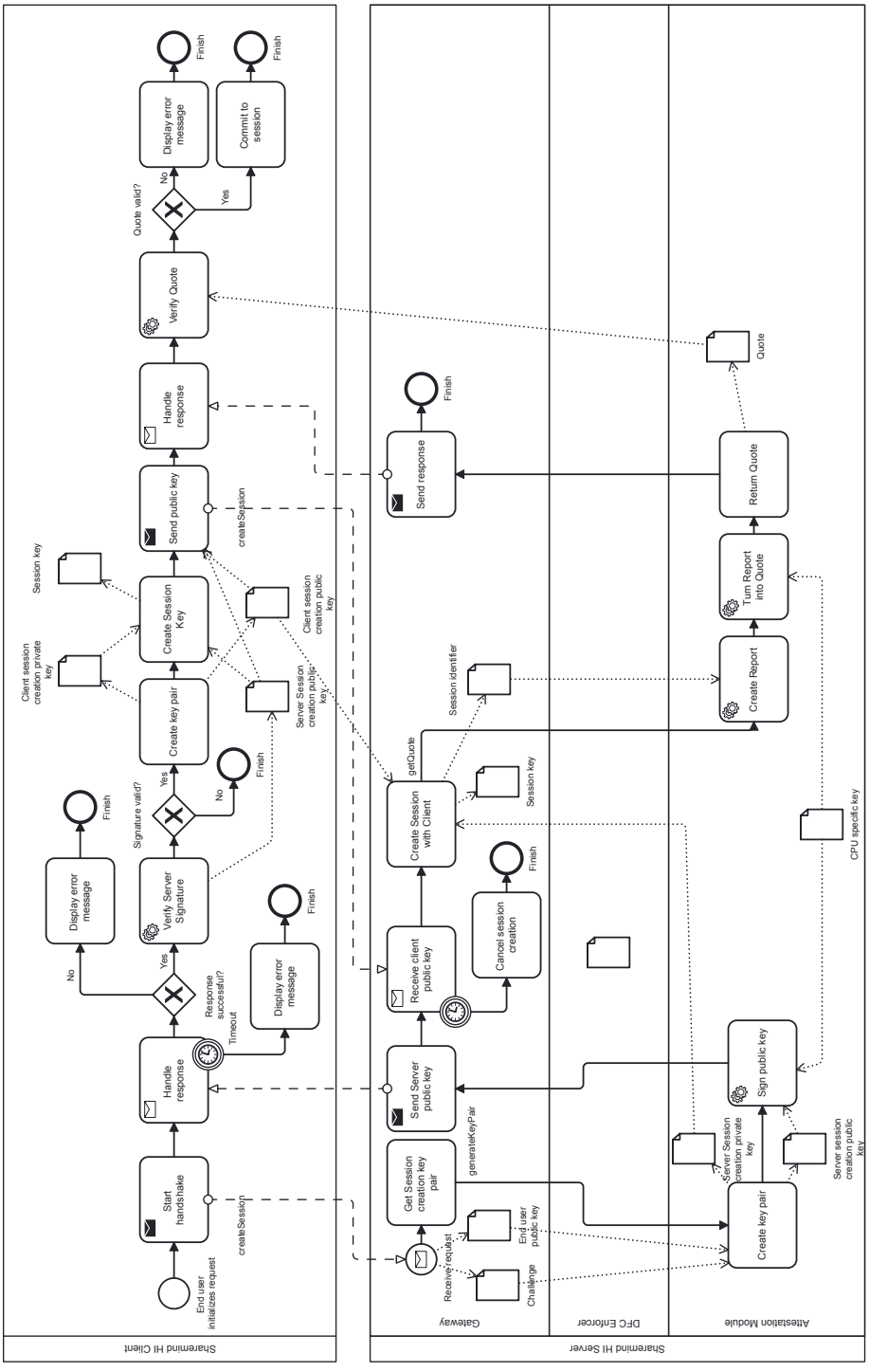

Figure 5. BPMN diagram of session creation process

Table 6. Data upload scenario description

Overview	Data Producer uploads data to Sharemind HI Server to make it available for analysis.	
System state	A session between Sharemind HI Server and Data Producer has been established.	
System environment	As usual	
External stimulus	Data upload request is made by Data Producer	
Required system response	Main success scenario	Data is saved as Data Entry in Persistent Storage, Client receives DEK.
	Session expired	Initialise new session brokering process, then repeat data upload.
	Unauthorised action Server/Persistent Storage timeout	Cancel data upload. Cancel data upload.

2. Gateway authenticates the user based on the session identifier. On failure, the Client will try to recreate the session.
3. Gateway uses DFC Enforcer's *isUserAuthorised* function to request authorisation of the user.
4. DFC Enforcer verifies that Client can upload data to given topic. On failure, a corresponding exception is returned to the client.
5. Gateway requests Persistence Controller to store the data using *storeData* function.
6. Persistence Controller generates DEK using nonce.
7. Persistence Controller generates eDEK using Master Key and DEK.
8. Persistence Controller constructs DataEntry and uploads it to the Persistent Storage.
9. Persistent Storage stores the Data Entry.
10. Persistent Storage responds to Persistence Controller with a success response.
11. Persistence Controller returns DEK to the Gateway.
12. Gateway returns the DEK along with a success message to the Client.

Data download is a process through which Data Consumers get access to analytics results. The process starts with Data Consumer creating a request for data with specific identifier. In the main success scenario Server responds with data entry of given id. If the Data Consumer's session can not be found based on the Server-side, then a response requiring reestablishment of the session is sent. If the Data Consumer does not have permission to download this data, a corresponding exception message should be returned,

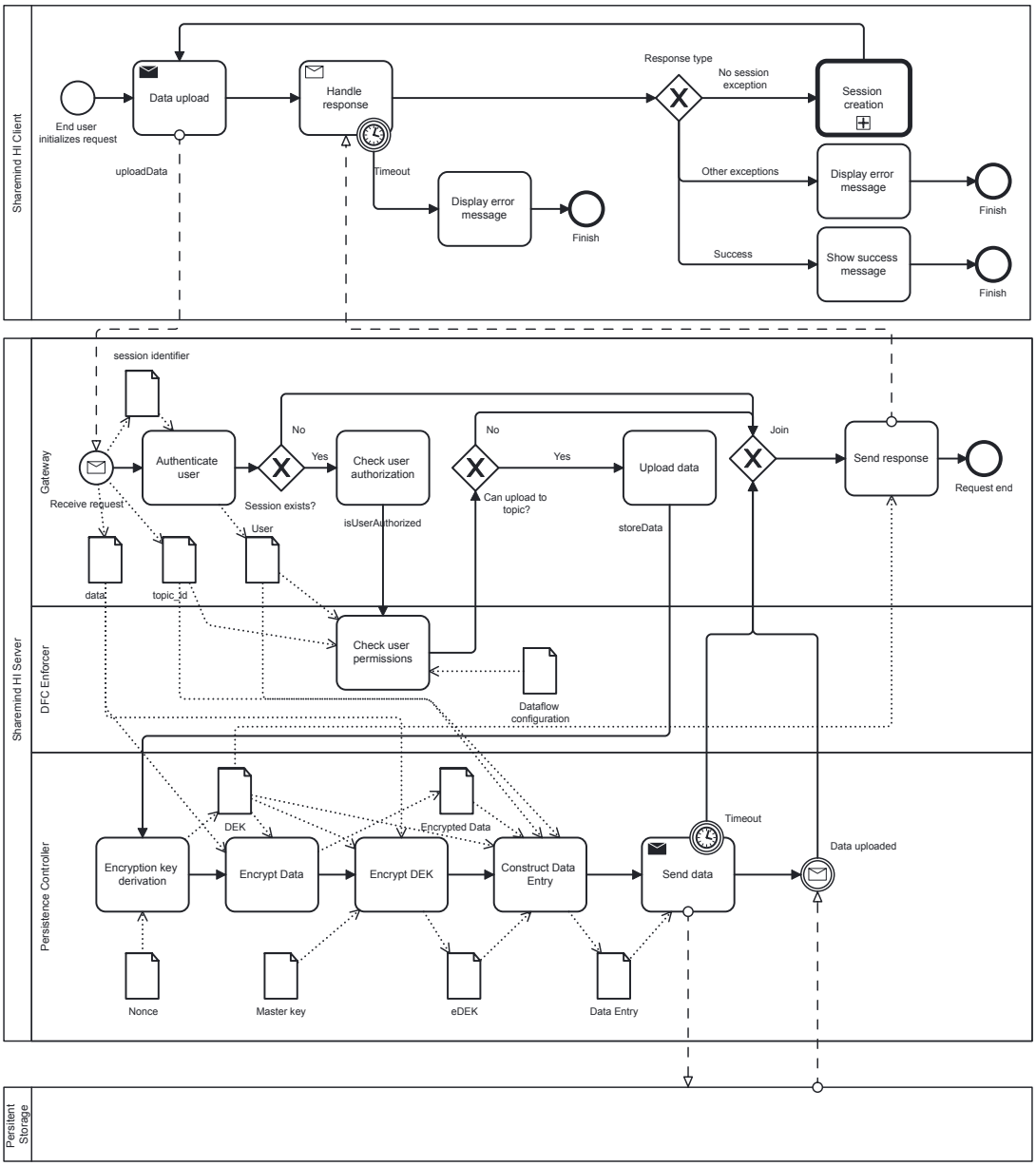

Figure 6. BPMN diagram of data upload process

and the process should stop. If the Server or Persistent Storage takes longer, the process is stopped, and a timeout exception should be shown in the Client application. See Section 4.2.1 for a summary of the scenario.

Using the constructed architecture, the scenario can be completed as follows (also see Section 4.2.1):

Table 7. Data download scenario description

Overview	End user marked in the DFC as a Data Consumer in a topic downloads data from said topic.	
System state	End user has established trust or trusts Sharemind HI instance.	
System environment	As usual	
External stimulus	Data download request is made from the end user	
Required system response	Main success scenario	Data is downloaded
	Session expired	Initialise new session brokering process, then repeat data download
	Unauthorised action	Cancel data download
	Server/Persistent Storage timeout	Cancel data download

1. Client sends a request to Server to fetch data with certain identifier using *dataDownload* function.
2. Gateway authenticates the user based on session identifier. On failure, the Client will recreate the session.
3. Gateway uses DFC Enforcer's *isUserAuthorised* function to request authorisation of the user.
4. DFC Enforcer verifies that Client can download given Data Entry. On failure, a corresponding exception is returned to the client.
5. On success, the Persistence Controller fetches data from Persistent Storage using retrieveData.
6. Persistence Controller decrypts the DEK, from the eDEK and Master key.
7. Persistence Controller decrypts data using DEK.
8. Persistence Controller sends decrypted data to Gateway.
9. Gateway sends decrypted data as response to the Client.

Run task. Run task scenario describes how Tasks are run on the server. The request starts when an end user creates task run requests through Sharemind HI Client. The main success scenario is that the server runs the Task and the Tasks returns a Response to the Task Runner. If Task Runner's session is expired, a session recreation is requested before rerunning the Task. If Task Runner aims to run a Task for which he does not have permission, a corresponding exception must be thrown. Similarly, Task run should be cancelled when a request to the Server times out or an unhandleable exception occurs in the Task. The scenario is summarised in Table 8.

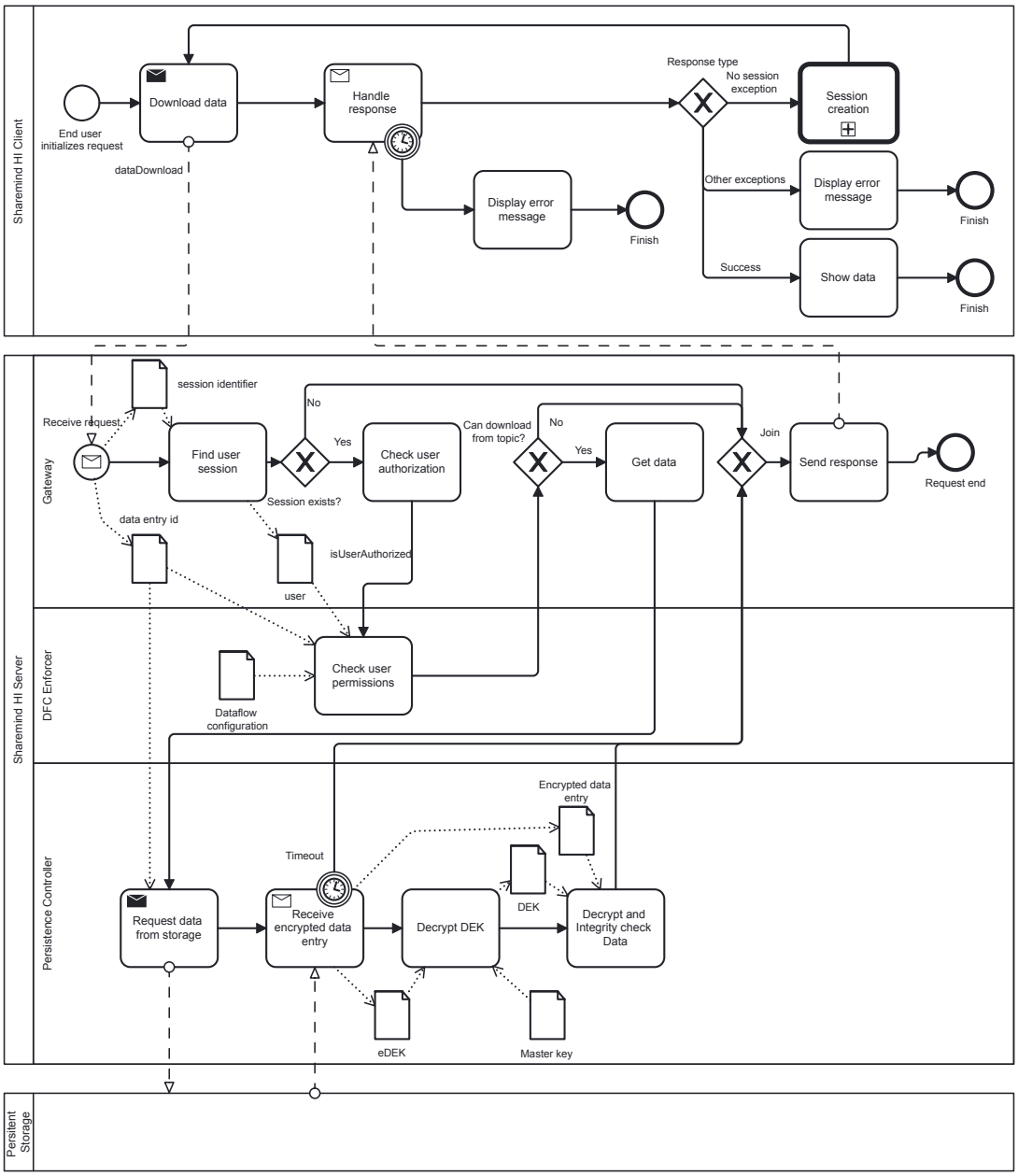

Figure 7. BPMN diagram of data download process

Using the constructed architecture, the scenario can be completed as follows (also see Fig. 8):

1. Client sends a request to Server to run a task using *runTask* function.
2. Gateway authenticates the user using a session identifier. On failure, its response re-

Table 8. Task run scenario description

Overview	End user marked in the DFC as a Task Runner runs analysis in the Server.	
System state	End user has established trust or trusts Sharemind HI instance.	
System environment	As usual	
External stimulus	End user starts Task run.	
Required system response	Main success scenario	Task is run
	Session expired	Initialise new session brokering process, then repeat task run
	Unauthorised action	Cancel task run
	Unhandleable exception in Task	Cancel task run
	Server timeout	Cancel task run

quires recreating the session, which the Sharemind HI Client handles automatically before resubmitting the data download request.

3. Gateway uses DFC Enforcer's *isUserAuthorised* function to request authorisation of the user.
4. DFC Enforcer verifies the client's authorisation to run the task. On failure, a corresponding exception is sent to the client.
5. On success, Gateway requests Task Manager to run the task using Task Manager's *runTask* function.
6. Task Manager requests the Task to run the analysis by calling Task's *run* method. .
7. The Task performs the analytics, reading and writing data to persistent storage over file system using FsInterface and eventually sends a response to Task Manager
8. Task Manager relays Tasks response to Client over Gateway.

Verify DFC Each version of DFC must be verified by all Enforcers before it goes active. Before activated, they are in a staging state. This scenario describes the process of querying DFCs and publicly stating trust or distrust in a given staging DFC. In the main success scenario, the verification result is stored in the Persistent Storage and a success response is returned to the Enforcer. If Enforcer gives an invalid verification result or attaches a valid verification result to an invalid or expired staging DFC, the process stops with an exception. Similarly, when a regular End user, who is not marked as an Enforcer in the DFC, attempts to verify the DFC, the process is stopped. The scenario is summarised in Table 9.

Using the constructed architecture, the scenario can be completed as follows (also see Fig. 9):

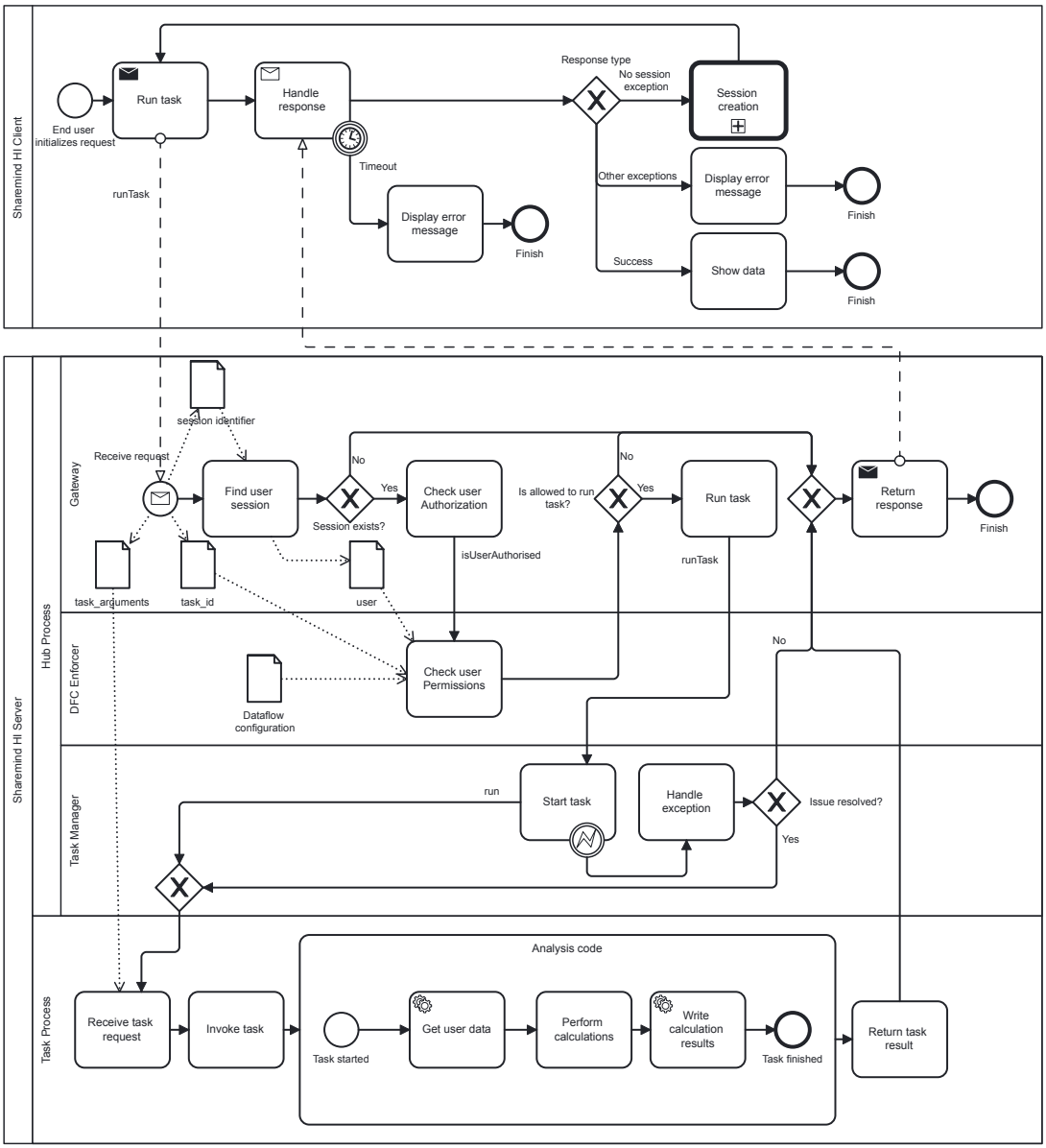

Figure 8. BPMN diagram of task run process

1. Enforcer requests Staging DFC from Persistent Storage.
2. Enforcer verifies the DFC.
3. Enforcer sends the verification result using Sharemind HI Client library to Sharemind HI Server.
4. Client sends the request to Gateway using *verifyDFC* function.

Table 9. Verify DFC scenario description

Overview	Enforcers can verify staging DFC.	
System state	Sharemind HI Server application is running.	
System environment	As usual.	
External stimulus	Client initialises the connection with Sharemind HI Server.	
Required system response	Main Success Scenario	Enforcer's verification result is added to the staging DFC.
	Enforcer uploaded DFC staging DFC is not valid	Cancel DFC verification.
	Enforcer gives invalid verification response	Cancel DFC verification.
	Unauthorised action	Cancel DFC verification.

5. Gateway authenticates the user based on the session identifier.
6. Gateway request authorisation check of the user form DFC Enforcer using *isUserAuthorised* method.
7. Persistence Controller either downloads the staging DFC to the database or returns it from the cache.
8. The staging DFC is sent to the Client.
9. Enforcer can give approval or deny the version of DFC.
10. Client transmits the verdict to Server.
11. Persistence Controller stores the verdict in Persistent Storage. The integrity of the verdict is protected using MAC.
12. Once Persistent Storage has confirmed the storage of verdict, the Persistence Controller can relay the success response to the Client.

Upgrade DFC As the business environment may change, so it is also necessary that the DFC be upgraded accordingly. The process is started by the Administrator, who provides a new DFC. In the main success scenario, the DFC is approved and upgraded. If, however, any of the Enforcers reject the DFC, the DFC is discarded. Similarly, DFC is discarded if errors are detected during early syntax check. See Table 10 for scenario summary.

Using the constructed architecture, the scenario can be completed as follows (also see Fig. 10):

1. Client sends a request for uploading a staging DFC using *uploadDFC* function.
2. Gateway forwards the request to DFC Enforcer using *addStagingDFC* function of DfcInterface.

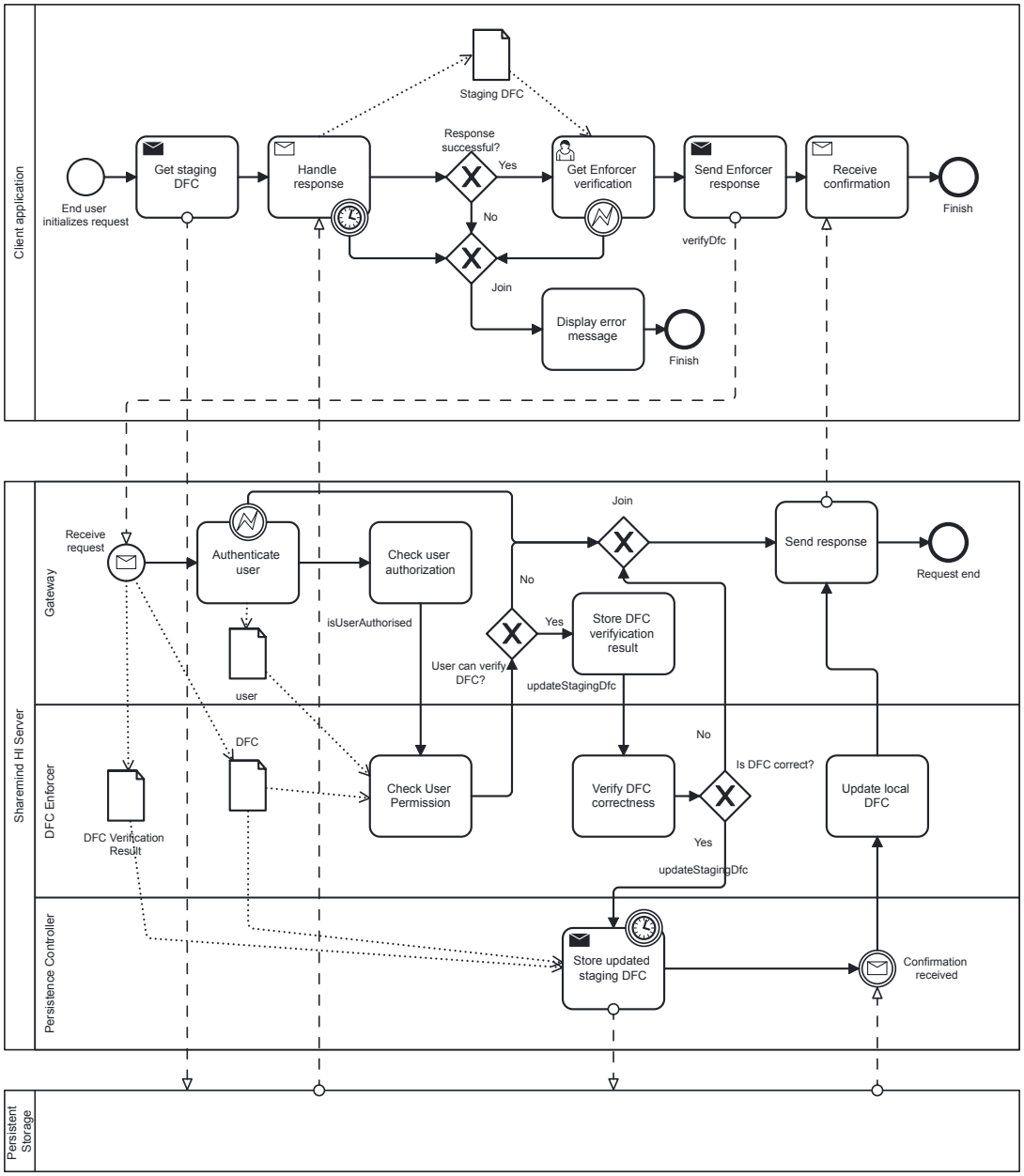

Figure 9. BPMN diagram of verify DFC process

Table 10. DFC upgrade scenario descriptions

Overview	Process describing how DFC can be updated.	
System state	Sharemind HI Server application is running.	
System environment	As usual.	
External stimulus	Administrator uploads a new DFC.	
Required system response	Main Success Scenario	DFC is upgraded, server is restarted in recovery mode.
	Enforcer disapproves DFC	DFC is disqualified.
	DFC is not syntactically correct	DFC is disqualified. Problem is given.

3. DFC Enforcer controls the syntax of uploaded DFC. If errors are found, an exception is thrown, else the DFC is stored using *addStagingDFC* function of PsInterface.
4. Persistent Controller stores the DFC in Persistent Storage.
5. Once Persistent Controller receives confirmation from Persistent Storage, a response is sent to the DFC Enforcer.
6. DFC Enforcer adds DFC to local DFC History.
7. DFC Enforcer returns through the Gateway a success response to the Client.
8. Client informs Administrator of DFC being uploaded.
9. Administrator informs Enforcers through out-of-bands medium of the staging DFC.
10. Enforcers either unanimously accept the DFC or the DFC is discarded.
11. If the staging DFC is accepted, Administrator marks the DFC in Persistent storage as active.
12. Administrator then uses Client to restart the server using *upgradeDfc* function.

Server recovery Server recovery scenario describes situation where Administrator starts Sharemind HI Server after shutdown. The process starts at Client library after the request from Administrator has been created. In the main success scenario the Server is booted up, the DFC History and Master Key are restored. If DFC History is tampered, the Server recovery stops. Notably, there is no protection against replay attacks - Persistent Store can store a history of valid DFC Histories and at the recovery give any one of these valid DFC Histories. See Table 11 for summary of the scenario.

Using the constructed architecture, the scenario can be completed as follows (also see Fig. 11):

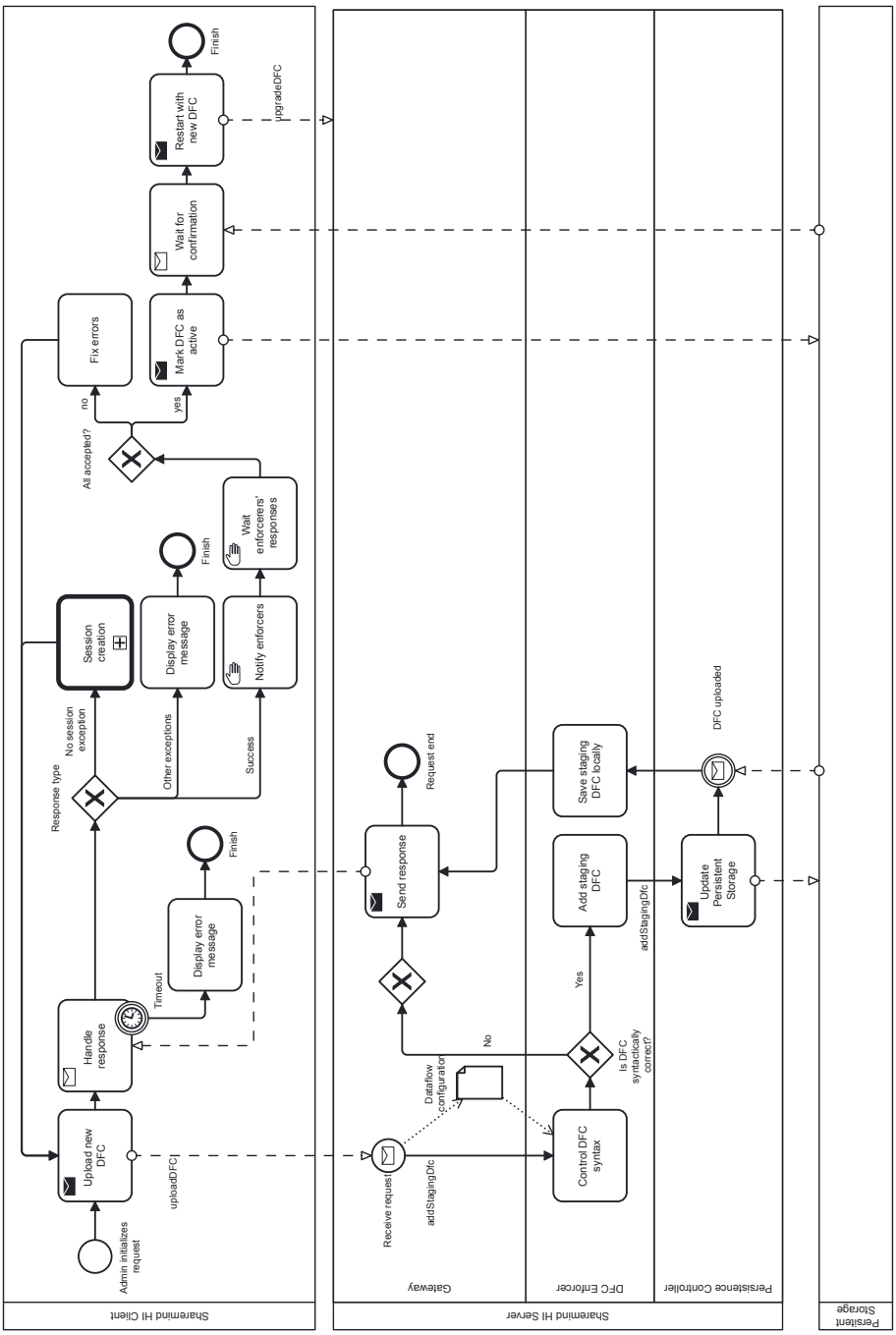

Figure 10. BPMN diagram of upgrading DFC

Overview	Administrators starts recovery of Sharemind HI Server after reboot.	
System state	System is started, DFC History and Master Key are missing.	
System environment	If server failed due to hardware failure, the failing parts are replaced.	
External stimulus	Administrator starts server recovery.	
Required system response	Main success scenario	Server is restarted and the state is recovered.
	Tampered DFC	Stop recovery process. Show corresponding Exception.
	Bad KBS Quote	Stop recovery process. Show corresponding Exception.

Table 11. Server recovery scenario descriptions

1. Client sends a request to Gateway to start the recovery process using *startRecovery* function.
2. Gateway sends a request to DFC Enforcer to fetch the DFC history using *fetchDfcHistory* function of DfcInterface.
3. DFC Enforcer forwards the request to Persistence Controller using *fetchDfcHistory* function of PcInterface.
4. Persistence Controller fetches the DFC History from Persistent Storage.
5. Persistence Controller verifies the integrity of DFC History. If a modification is detected, the DFC History is discarded and the process stops with an exception.
6. DFC Enforcer starts Master Key retrieval with the latest non-staging DFC in DFC History using getMasterKey function.
7. Persistence Controller acts as Relying Party and requests Quote from the KBS.
8. KBS sends its Quote to Persistence Controller.
9. Persistence Controller requests Attestation Module to verify the KBS Quote using *verifyQuote* function.
10. Attestation Module verifies KBS Quote using Attestation Provisioning Service.
11. Persistence Controller request Master Key from KBS.
12. KBS acts as a Relying Party and requests Quote from Sharemind HI Server.
13. KBS gives a Master Key against the Quote.
14. KBS returns the key to the Persistent Contoller.
15. Persistent Controller returns a success response back to Client.

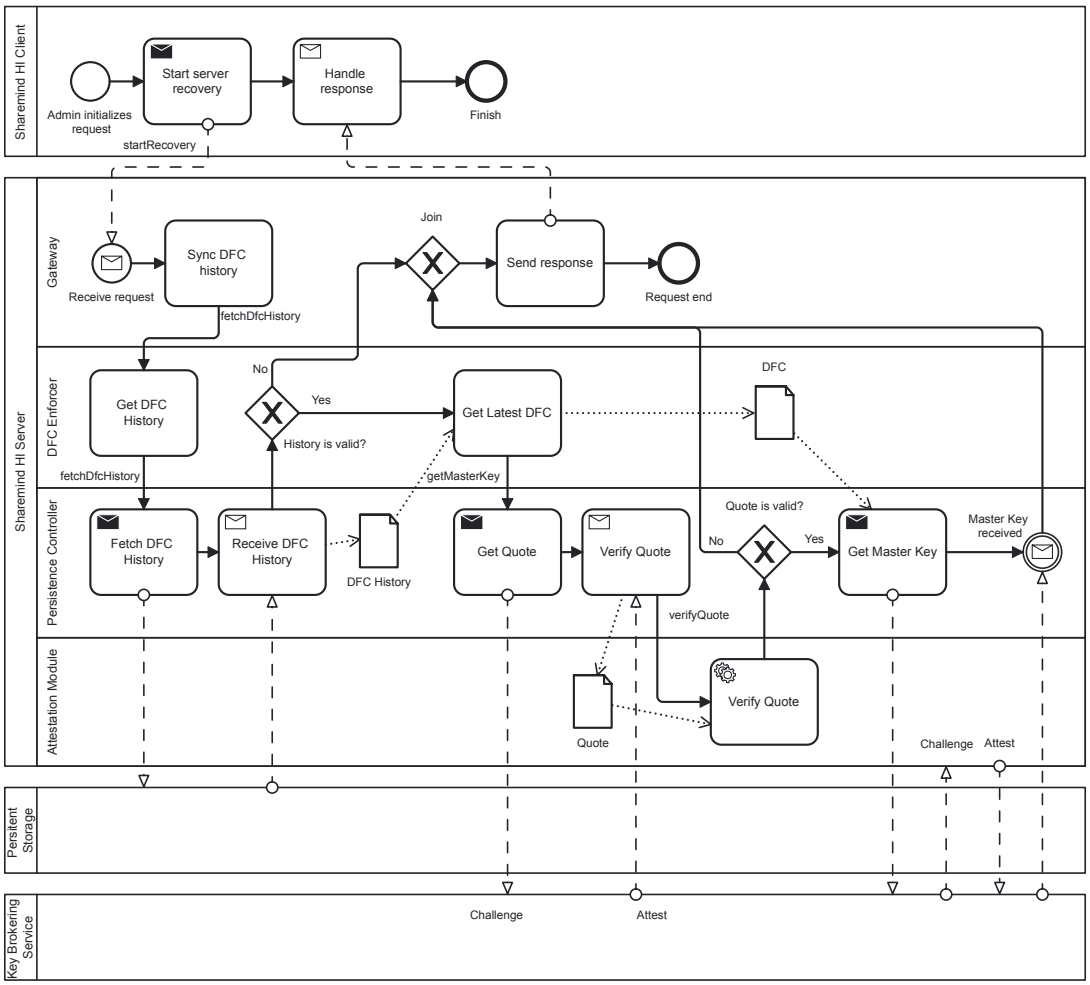

Figure 11. BPMN diagram of server recovery

4.2.2 Security Scenarios

The main security goal of Sharemind HI is to protect user data against unauthorised access [7]. In the White Paper of SGX-based Sharemind HI a total of 12 threats are highlighted distributed among various Threat agents. It then states requirements to negate these threats and then lists technical and organisational controls that fulfil the requirements. By providing secure data persistence using Persistent Storage Provider and KMS, and expanding the TEE size to a VM, a new Threat agent and additional attack methods for existing Threat agents were created. Going over different software and network attacks given in Common Attack Pattern Enumerations and Classifications (CAPEC) [35], a set of new potential attacks were detected and described using ISSRM meta-model.

Malicious Task gets deployed in production Verifying code correctness is difficult and through clever stenography techniques Task Developers could hide malicious functionality to bypass enforcers' reviews. This scenario considers the situation, where a malicious Task is in the Sharemind HI Server (Enforcers have accepted the DFC containing malicious Rask) and we look at different attacks it can perform. While the confidentiality of topic data intended for the Task is broken as the malicious Task can extract it in the response to Task Runner, it is required that the malicious Task could not access data in other topics. In other words breaches should be isolated.

We found two attacks that a malicious could perform - Port Stealing (CAPEC-151: Identity Spoofing), and Privilege Escalation and Abuse (CAPEC-233, CAPEC-122). Port Stealing is an attack, where one Task(T1) binds to port meant for another Task(T2). As a result, when the Task Manager wants to send a TaskRun request with Confidential task arguments to T2, T1 intercepts the message. To extract the data, malicious Task can write the confidential parameters to a topic the attacker has access to, or if networking is not closed, the malicious Task can send the data to attacker defined address. The attack negates confidentiality of task parameters and damages availability of other tasks as they can no longer start the HTTP server due to port already being in use. The attack exploits lack of authentication methods and therefore its closest counterpart in CAPEC database is Identity Spoofing attack (CAPEC-151).

To protect against this attacker, Sharemind HI Server must assign and enforce the ports of the Tasks it hosts. This could be done in Linux by running each Task under different user and then creating strict firewall rules that filter out all other traffic through other ports. Note that, this still leaves the possibility to damage availability of other Tasks. To solve that issue, Task Manager could preemptively kill Tasks holding multiple ports or alternatively if a Task reports port-in-use exception, Task Manager could kill the occupying processes. The risk is summarised in Table 12.

While the task process by default should not have permissions to read memory of other processes as a user space program, the attack surface of an operating system is large and therefore weaknesses might slip in, which could be used to gain these permissions. The privilege of reading memory of other processes could be used to get access to the master key, which could then be extracted to break confidentiality of Data Producers' data. Additionally owning Master key further attacks on integrity are possible by colluding with Persistent Storage. The associated risk is summarised in Table 13. To tackle this attack the tasks would have to be run in sandboxed environments. An established way to sandbox applications in Linux is using seccomp [33]. The library enables management of syscalls that a process can do, thereby limiting attack surface for privilege escalation attacks. seccomp is supported in systemd, a service manager in Linux distributions, which provides an accessible interface for implementing policies at service level [10]. We propose to run Tasks as systemd services and use its sandboxing methods to limit the attack surface.

Table 12. Description of R1

Name	R1 - Port Stealing
Threat agent	Task code developer
Attack method	1. Malicious Task is deployed in the Server. 2. Malicious Task systematically connects to all available ports. 3. Malicious Task captures all incoming Task Run requests. 4. Malicious Task reads confidential parameters from the requests. 5a. Malicious Task writes confidential parameters to a topic, which is available to the attacker. 5b. Alternatively Malicious Task returns the Confidential data to Task Runner with whom the malicious Task develop has colluded with. 5c. Alternatively Malicious Task extracts data through I/O devices, e.g sends the data over network.
Vulnerability	CWE-287: Improper Authentication - No authentication is implemented for connecting to ports.
Impact	Negation of confidentiality in all messages sent between Task manager and Task Processes.
IS asset	R1.1, R1.2 - In Task run process, intended Task's functionality is overridden by Malicious Task, see Fig. 19
Business asset	R1.1 - User arguments (C) R1.2 - Task response (I)
Severity	Medium
Difficulty	Medium
Risk treatment	Reducing the risk through technical controls - firewall rules.

Malicious Persistent Storage Administrator tries to exploit his privileges. Persistence Storage Administrator has read/write access to DA applications persistent data as well as all control over the network of the Persistent Storage. The security goal is to protect confidentiality and integrity of all data entries, as well as integrity of DFC history. We identified total of 3 attacks against this goal: Object Injection, Resource Location Spoofing and Data Reconstruction Attack.

As the Persistent Storage Administrator has write permissions, he can modify data such that it can even trigger remote code execution during deserialisation. The described attack is called Object Injection (CAPEC-586) in CAPEC attack library. In TDX-based Sharemind HI, the deserialisation occurs in the Persistent Storage Controller, which as a system component has higher level privileges. In worst case, the injected code could extract Master Key from the Server leading to loss of confidentiality of user data. Having the Master Key, the Administrator can modify MAC protected objects

Table 13. Description of R2

Name	R2 - Privilege Escalation and Abuse (CAPEC-122)
Threat agent	Task code developer
Attack method	1. Malicious Task process is run inside Sharemind HI Server TEE. 2. Malicious Task manages gain privileges he is not supposed to have. 3. Malicious Task process identifies a process that runs alongside in the Sharemind HI Server. 4. Malicious Task process reads memory of target process. 5. Malicious Task process locates and reads confidential user data in the memory.
Vulnerability	CWE-269: Improper Privilege Management - Task process should not have permission to read memory of other processes. CWE-653: Improper Isolation or Compartmentalisation - Task process should not be able to detect other processes running in the system.
Impact	Confidentiality and integrity of all business assets are negated, Sharemind HI Server process flow disturbed.
IS asset	Sharemind HI Server
Business asset	All business assets that move through Sharemind HI Server application.
Severity	Extremely high
Difficulty	Very high
Risk treatment	Reducing the risk - sandboxing Transferring the risk - enforcers of the DFC

in Persistent Storage and calculate a new MAC, leading to negation of integrity. To avoid Object Injection, the verification of the data object must be done before attempting to deserialize it. This means that in TDX-based Sharemind HI the communication with Persistent Storage must happen over Persistence Controller to limit attack surface. Additionally, existing and established crypto libraries such as libsodium [28] should be used Authenticated Encryption with Associated Data (AEAD). For summary of the risk see Table 14.

Secondly, the Persistent Storage Administrator can perform a Resource Location Spoofing attack. When a Task request certain data entry, the Persistent Storage Administrator can configure his service to respond with another valid data entry from other topic. As the data entry is valid Persistent Controller deserialises, decrypts it and sends it to the Task. However, the Task might not be allowed to access this data, thereby potentially breaking confidentiality of data passed to the Task. On the other hand, the integrity of the

Table 14. Description of R3

Name	R3 - Object Injection (CAPEC-586)
Threat agent	Persistent storage provider
Attack method	1. Data is uploaded to persistent storage. 2. Sharemind HI requests data from persistent storage. 3. Persistent storage carefully constructs message that during deserialisation causes malicious code to be run. 4. Persistent storage controller deserialises the data and causes remote code to be run in the server.
Vulnerability	CWE-502: Deserialization of Untrusted Data - Persistent storage controller implementation deserializes data before verifying its integrity.
Impact	Deserialization of untrusted Data can lead to remote code execution [35].
IS asset	R3.1 - Persistence Controller in Fig. 18 R3.2 - Task process in Fig. 19 R3.3 - Persistence Controller in Fig. 22 R3.4 - Client application in Fig. 20
Business asset	R3.1 - Data Entry (I, C) R3.2 - Data Entry (I, C) R3.3 - DFC (I) R3.4 - DFC (I)
Severity	Very High
Difficulty	Medium
Risk treatment	Reducing the risk - established crypto library and limiting attack surface.

Task output data can be considered to be broken as the input was maliciously altered. To resist against this attack, identifiers of data stored in Persistent Storage must be assigned by unique and under integrity protection. When data is read from the Data Storage, Persistence Controller must verify that the received data identifier is that of the requested data identifier.

In the context of this work Data Reconstruction Attack is an attack foremost against integrity of the datasets. Persistent Storage Administrator can carefully construct a dataset by removing or hiding entries. This means that he can create either biased or one-row datasets. When a Task requests data for its analysis, the Persistent Storage will respond with the modified dataset, which means that the result of the computation is also influenced by it. In addition to impacting the integrity of the dataset, the attack could put confidentiality of the Data Producer's data in risk if the Administrator can

Table 15. Description of R4

Name	R4 - Resource Location Spoofing (CAPEC-154)
Threat agent	Persistent storage provider
Attack method	1. Sharemind HI requests list of available data in persistent storage. 2. Persistent storage provider creates a map of data entries with each entry containing identifier and a URI. 3. Persistent storage provider manipulates the list by replacing URI of one data entry with URI of an another. 4. Persistent storage provider returns the list to Sharemind HI Server. 5. Sharemind HI requests data as per URI matching the identifier from persistent storage. 6. Unexpected data is returned to Sharemind HI Server, which might be passed to task processes, which do not having the permissions to operate on this data.
Vulnerability	No related vulnerability.
Impact	The integrity and possibly confidentiality of business asset is negated
IS asset	R4.1 - Persistence Controller in Fig. 18. R4.2 - Task Process in Fig. 19. R4.4 - Persistence Controller in Fig. 22 R4.4 - Client application in Fig. 20
Business asset	R4.1 - Data Entry (I, C) R4.2 - Data Entry (I, C) R4.3 - DFC (I) R4.4 - DFC (I)
Severity	Medium
Difficulty	Medium
Risk treatment	Reducing the risk - unique and authenticated identifiers, and limiting attack surface.

access the result of the computation. The risk of this attack can be mitigated by using Circular Message Authentication Code (CMAC) as proposed by Silverio et al. in [34]. By adding additional field called CMAC to Data Entry, which is calculated based on the MAC of previous Data Entry and MAC of current Data Entry, there forms a chain from the first data entry to the last one as seen in Fig. 12. The CMAC of the first Data Entry is recalculated during each data upload to contain the MAC of the uploaded Data Entry, thereby turning the chain into circle. If Persistent Storage obstructs sharing certain

Data Entries, the omission can be detected. CMAC does not protect against replay attack, meaning that first uploaded Data Entries are still vulnerable to Data Reconstruction Attack. To protect these Data Entries additonal rules could be set for running Tasks, that forbid running them when less than configured amount of Data Entries are present.s The overview of the risk is given in Table 16.

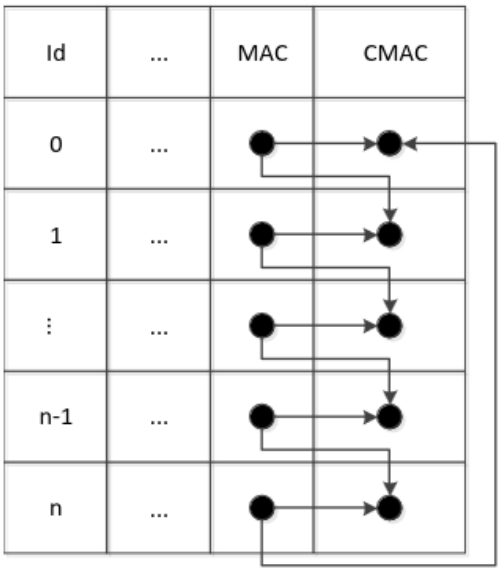

Figure 12. Visualisation how CMAC is used to protect integrity of the data table [34]

4.2.3 Ease of Development Scenario

We showed that the only requirement for the programming language is that it must be able to start an HTTP server in order to receive *run* invocations. In order to fetch Data Entries, file system operations should also be available in the programming language. These are very lenient requirements, and in practice, it means that any programming language can be used. The focus of this chapter is extending the argument and how ease of development increases in the scenario where programming language and libaries are not restricted.

Firstly, C/C++ does not have built-ins required for the compiler to detect unsafe memory operations. This means that the developer requires a lot more effort to write and review code. Meanwhile, there are languages such as Rust and Go that, through additional language features, can find these problems automatically at compile time, reducing the load on the developer. This issue was highlighted by White House's report [14] in the context of security, however it is equally valid from ease-of-development

Table 16. Description of R5

Name	R5 - Data Reconstruction Attack
Threat agent	Persistent storage provider
Attack method	1. Task process requests collection of data entries from persistent storage provider. 2. Persistent storage crafts a subset of data and passes it to the task.
	3. Task process performs some aggregate analytics on the dataset.
	4. Task process obtains biased result from the analytics.
Vulnerability	No related vulnerability.
Impact	Integrity of the dataset is negated. Confidentiality of Data Entries is in risk.
IS asset	R5 - Task process in Fig. 19
Business asset	R5 - Data Entry (I)
Severity	High
Difficulty	Medium
Risk treatment	Reducing the risk - CMAC

perspective. Using memory-safe language eases the development of Sharemind HI because the Sharemind HI developer does not have to verify the code himself.

Secondly, in the data science domain, generally, higher-level programming languages such as Python and R are used. For example, at the time of writing, there were a total of 1804 courses under the Data Science category. Searched by skills, only 2 of those courses were tagged with C++. On the other the same number was 406 for Python and 157 for R [37]. This indicates that people in data science domain have more experience with these programming languages. Therefore, having the ability to choose the programming language makes Task developer's job easier because they can use the language they are familiar with.

Thirdly, commonly a data analysis already exists, which requires additional security protection from cloud provider. In the proposed architecture, to port these solutions to Task, the I/O interfaces need to be refactored such that analysis data is read from files, all results are written to files, and an HTTP must be set up. In the old architecture, the changes would have required rewriting the application in C/C++. If the program was already in C language, then the previous step could be avoided, however library dependencies would have to be still checked. If any of the used libraries is not in SGX SDK's supported list [18] the Task developer would have to find an alternative.

4.3 Measuring Overhead

While compared to other PETs that offer data-in-use protection, TEEs have much lower overhead [36], a recognisable performance penalty still exists. Intel's report on Intel TDX performance considerations brings out three culprits: transitions in and out of trusted domain require extra steps compared to entering and exiting VMs; cache misses in TEE workflows have a higher cost as all reads from memory must include decrypting its contents and verifying integrity before actually being able to use it; I/O calls are implemented using a shared unencrypted memory range between IO device and the TD, which means that the TEE has to encrypt all data it takes from the shared memory and decrypt all data it wants to make available.

To measure the effect of these sources of overhead in practice, we prototyped a system mock-up that followed the proposed architecture and 2 different performance test flows. Both tests were run in TD and regular VM. The prototype code is written in C++, the persistent storage is a PostgreSQL database. The mocked Sharemind HI Server was deployed in Google Cloud as a c3-standard-4 VM instance. According to Google Cloud's documentation [13], the VM instance has Intel's 4th Generation Scalable Xeon, with 4 hardware threads or logical cores and 16 GB of RAM. Whether the VM is deployed as a regular VM or TD could be configured at the VM creation. The hardware was physically located in the West-Europe.

The test flows were designed to capture different parts of data lifecycle in Sharemind HI's data analysis model. Processes involving data are always between the Server and Client, Server and Persistent Storage or inside the Server. The first performance test flow aims to measure the overhead of the Client-Server data transit and the second performance test flow aims to measure the overhead of Database-Server data transit. The part measuring the overhead of data inside the server was left out of scope due to limited time. Additionally, Task code is highly specific to the project, and therefore, tests would likely not be more accurate than Intel's 5% overhead found in [22]. The number was obtained from running 5 different CPU and memory-intensive processes.

PT1. Client - Task roundtrip. The test measures the time it takes for the Client to send a message to Task and get a response. Furthermore, additional measurements are taken in Gateway and Task during entrances and exits, see Fig. 13. The test mocks situations, where a Client has already established a trusted session with the Server and sends request to the Server. The purpose of the test is to measure the overall overhead of the Client-Task interaction latency and the overhead inside the Server when message is passed to Task. To see the effect, which task arguments size has on the overhead, we repeat the test with argument sizes increasing tenfold at each call, ranging from 1 byte to 10 megabytes. In total for each argument size, we ran the test 4000 times. The code is provided in Listings 1 to 3.

The results of the first test flow show no or minimal overhead in all places of measurement (Task, Gateway and Client). In Gateway, there seem to be higher overhead

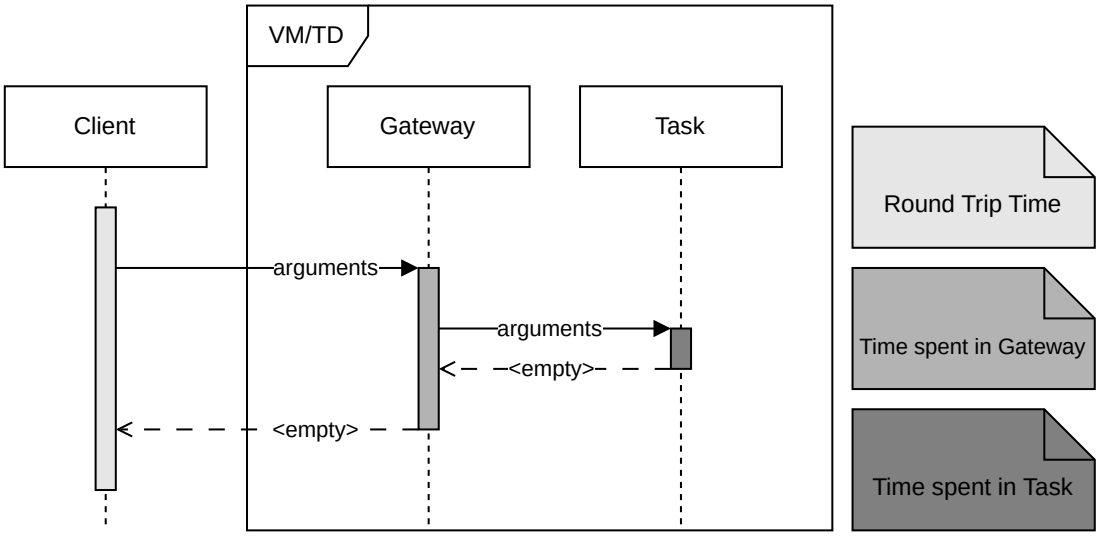

Figure 13. UML sequence diagram of the PT1. Different colours of grey signify the part of the workflow, which were measured during the test flow

numbers than in Client and Task, though the difference is small and could be due to high standard deviation. In terms of data size, it seems TD seems to perform better with larger bitstrings, while with smaller data sizes VM seems to have an advantage. For example, with bitstring size of 1 MB there was a negative overhead of -10.27% measured in Client. The results are displayed in Tables 17 to 19

Table 17. Time spent in Tasks measured in PT1. The table displays absolute runtime of the same workload in different execution environments and relative overhead of TD.

	TD		Regular VM		
Bitstring size (B)	Mean average (ms)	Standard deviation (ms)	Mean average (ms)	Standard deviation (ms)	Mean overhead (%)
10^0	0.0187	0.0058	0.0186	0.0039	0.4103
10^1	0.0187	0.0046	0.0187	0.0042	0.0107
10^2	0.0189	0.0041	0.0189	0.0041	0.1455
10^3	0.0189	0.0041	0.0189	0.0038	0.5105
10^4	0.0259	0.0050	0.0256	0.0049	1.0892
10^5	0.0789	0.0123	0.0795	0.0128	-0.7500
10^6	0.8304	0.0758	0.8778	0.0765	-5.3954
10^7	9.9438	1.1535	9.9992	1.1664	-0.5547

PT2. Server - Persistent Storage The second test is initialised in the Server, which then requests data from the Persistent Storage. The persistent storage fetches data entries and responds to the Server. The test aims to measure, the overhead of reading data from Persistent Storage and see the impact of size of the requested data on the overhead. The Persistent Storage used in the test contains random bitstrings starting from one byte and

Table 18. Time spent in Gateway as measured in PT1. The table displays absolute runtime of the same workload in different execution environments and relative overhead of TD.

	TD		Regular VM		
Bitstring size (B)	Mean average (ms)	Standard deviation (ms)	Mean average (ms)	Standard deviation (ms)	Mean overhead (%)
10^0	0.339	0.039	0.331	0.040	2.507
10^1	0.340	0.032	0.330	0.030	3.005
10^2	0.340	0.033	0.331	0.037	2.463
10^3	0.339	0.028	0.331	0.029	2.408
10^4	0.346	0.029	0.336	0.032	3.133
10^5	0.414	0.036	0.406	0.036	1.994
10^6	1.203	0.081	1.244	0.085	-3.293
10^7	10.480	1.154	10.560	1.168	-0.762

Table 19. Total round-trip time measured in PT1. The table displays absolute runtime of the same workload in different execution environments and relative overhead of TD.

	TD		Regular VM		
Bitstring size (B)	Mean average (ms)	Standard deviation (ms)	Mean average (ms)	Standard deviation (ms)	Mean overhead (%)
10^0	41.09	4.61	41.21	6.03	-0.29
10^1	40.94	5.98	40.58	4.04	0.88
10^2	40.46	1.40	40.54	1.47	-0.21
10^3	40.65	4.38	40.66	1.77	-0.03
10^4	40.48	1.87	40.59	3.16	-0.27
10^5	46.82	13.14	48.01	14.38	-2.47
10^6	289.44	75.71	322.72	83.42	-10.31
10^7	2661.40	730.66	2710.12	823.27	-1.80

going up to 500 MB, increasing ten-fold at each interval. To get more robust results, the tables contained up to 1000 different bitstrings for each size, going lower as the length of the bitstrings increased. In one test case, all of the available data entries for specific bitstring size were queried and the average response time was found. For each bitstring size, the test was repeated for 250 times.

We did not detect any overhead for running processes in TD. In fact, for TDX-based workflows actually slightly outperformed their non-confidential counterparts for all bitstring sizes with results ranging from overhead of -1.05% to -0.21%. No patterns related to size were detected. The result are shown in Table 20 and visualized in Fig. 14 .

Evaluating the results. While cloud-based performance tests mimic the way the workloads are used in practice, their results are difficult to interpret as details necessary for analysis are abstracted away by cloud service providers. On one hand, network effects and multitenancy cause unstable results due to sharing resources with other tenants. On the other hand, when looking at Round-Trip-Time of PT1 and PT2 results,

Table 20. Total response time of a data request from database measured in PT2. The table displays absolute runtime of the same workload in different execution environments and relative overhead of TD.

	TD		Regular VM		
Bitstring size (B)	Mean average (ms)	Standard deviation (ms)	Mean average (ms)	Standard deviation (ms)	Mean overhead (%)
10^0	3.71	0.20	3.73	0.17	-0.74
10^1	3.72	0.20	3.74	0.17	-0.58
10^2	3.74	0.21	3.78	0.28	-1.05
10^3	3.96	0.24	4.00	0.22	-0.79
10^4	4.12	0.25	4.12	0.23	-0.06
10^5	5.54	0.31	5.59	0.32	-0.84
10^6	49.30	2.74	49.43	2.23	-0.26
10^7	124.03	14.43	124.87	16.28	-0.67
10^8	1387.27	165.9	1390.15	176.02	-0.21
$5 * 10^8$	7529.64	881.85	7548.95	939.90	-0.26

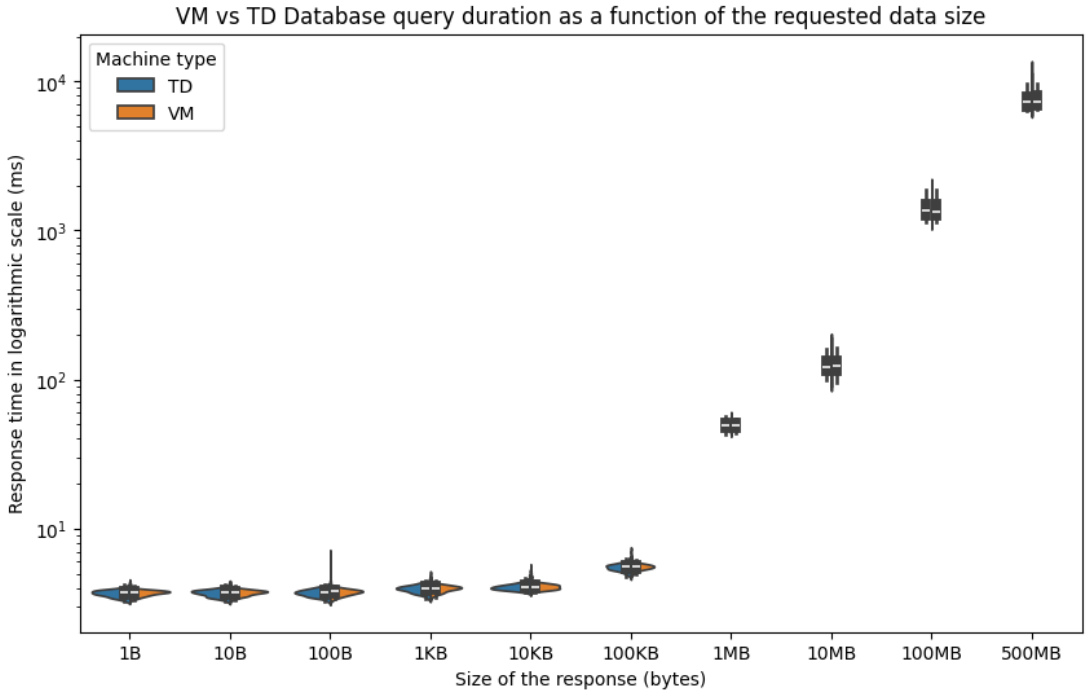

Figure 14. Violin plot of PT2 results, showing the similarity of VM-based and TD-based workflows in terms of performance.

there seemed to be a consistent negative overhead across all data sizes. One reason could be due to the price difference between running VMs and TDs in the cloud. Turning on confidential computing increases the price of computing [30]. Therefore, it could

be that there exists a lower demand for confidential computing compared to regular computing. Alternatively, Google Cloud could try to balance the performance of different computation environments.

4.4 Chapter Summary

The chapter focused on the last supporting research question "How does the new architecture solve the requirements?". In the first part we explained how the fulfilment of requirements will be verified. The main conclusion was that for performance requirements prototyping is more suitable as it gives more accurate estimations, and for all other requirements we use scenario based validation. In the second chapter we showed using scenarios-based evaluation that (1) TDX-based Sharemind HI is capable of solving core use cases of Sharemind HI, (2) in total 5 risks additional risks arise from using Intel TDX, all of which can be mitigated using technical controls (3) Intel TDX removes programming language and library constraints and provided 3 arguments why it simplifies development. In the third chapter, we demonstrated using prototpyed performance tests that TDX overhead in Sharemind HI related data flows in minimal.

5 Concluding Remarks

In the thesis we showed using an architecture how Sharemind HI could use Intel TDX to ease the development of DA applications. First we captured 5 requirements, which we wanted to verify using the architecutre. We then constructed the architecture description based on viewpoint-based framework. We then evaluated the architecture against the captured requirements.

5.1 Answers to Research Questions

The main research question of the thesis was stipulated as "How could Sharemind HI use Intel TDX to simplify task development?". The question was split into three supporting questions that when combined provide a sufficient answer to the original main research question.

What are the means of creating a verifiable model for TDX-based Sharemind HI? - The model of TDX-based Sharemind HI must be able to verify 5 requirements that arise from Intel TDX peculiarities and DA domain. The system was modelled using software architecture description from established framework that enables verifying requirements.

What is the architectural description of TDX-based Sharemind? - When looking at Sharemind HI as a black box, we see that it takes in requests from Sharemind HI Client and sends request to Persistent Storage and KBS during its lifecycle. The latter two components make it possible to persist data over restarts. Inside Sharemind HI Server there are seven functional elements distributed among a hub process and task processes. There are 6 different interfaces between functional elements that define the communication means between components.

How does the new architecture solve the requirements imposed on it? - We demonstrated using BPMN diagrams how the architecture is capable of solving core use cases of Sharemind HI. We used the diagrams to detect risks that lack of trusted persistence and merging execution environment of Tasks and the Server create, and proposed methods to mitigate these risks. The performance requirements were verified by developing system prototypes and performance tests that were then ran in TD and regular VM. The measured overhead was negligible.

5.2 Limitations and Future Work

There are three main limitations of the thesis, that could benefit from further work:

- "Bubbles don't crash" is an alleged quote by Bertrand Meyer [31] meant to invigorate healthy distrust in models. Models are not formal proofs, nor can they be tested against real-world scenarios. It is a well-noted criticism and is very

relevant in the context of this thesis. Even though established frameworks were used to capture the architecture, and we verified its completeness using 7 scenarios, there remains a risk that complications may arise during the implementation of the system. The complications are most likely to occur in Task sandboxing, KBS integration and development of the FUSE interface as they are completely new architecture-specific. The risk could be mitigated by first developing skeleton code to verify that it is possible to use them as described in the thesis.

- Cloud-based performance tests did not give conclusive results due to lack of transparency. For further work, we propose two paths. To better estimate the TDX-related, we recommend repeating the tests in a bare-metal local server. This would eliminate network and multitenancy effects, which otherwise overshadow the impact of different execution environments. Another interesting aspect would be to compare the performance of an application developed using SGX-based Sharemind HI and TDX-based Sharemind HI. This would give additional insight into the performance overhead of supporting processes such as encryption, policy enforcing and inter-process communication.

- Security aspects are only captured from the architectural angle and only captured select flows related to Task isolation and Persistent Storage. The remaining portion of the architecture and underlying TEE technology provide further attack vectors, which were left out of the scope of this thesis due to time constraints. This aspect deserves further investigation, especially considering that Sharemind HI promises to eliminate trust in cloud providers, a party with the means and motivation to perform hardware attacks.

References

[1] Michael Bartock et al. *Trusted cloud :: security practice guide for VMware hybrid cloud infrastructure as a service (IaaS) environments*. en. Tech. rep. NIST SP 1800-19. Gaithersburg, MD: National Institute of Standards and Technology (U.S.), Apr. 2022, NIST SP 1800–19. DOI: 10.6028/NIST.SP.1800-19. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-19.pdf (visited on 01/15/2024).

[2] Henk Birkholz et al. *Remote ATtestation procedureS (RATS) Architecture*. Request for Comments RFC 9334. Num Pages: 46. Internet Engineering Task Force, Jan. 2023. DOI: 10.17487/RFC9334. URL: https://datatracker.ietf.org/doc/rfc9334 (visited on 05/14/2024).

[3] Camunda. *The Universal Process Orchestrator*. en-US. URL: https://camunda.com/ (visited on 05/10/2024).

[4] Pau-Chen Cheng et al. *Intel TDX Demystified: A Top-Down Approach*. en. arXiv:2303.15540 [cs]. Mar. 2023. URL: http://arxiv.org/abs/2303.15540 (visited on 11/26/2023).

[5] Victor Costan and S. Devadas. "Intel SGX Explained". In: *IACR Cryptol. ePrint Arch.* (2016). URL: https://www.semanticscholar.org/paper/Intel-SGX-Explained-Costan-Devadas/a42e086f2382d518a0213879050e344539c2bcfa (visited on 12/15/2023).

[6] Cybernetica. *Sharemind HI Overview :: Sharemind Developer Zone*. URL: https://docs.sharemind.cyber.ee/sharemind-hi/4/general/sharemind-hi-overview.html (visited on 05/09/2024).

[7] Cybernetica. *Sharemind HI White Paper*. en. Jan. 2021. URL: https://cyber.ee/uploads/sharemind_hi_white_paper_ec24e8189a.pdf (visited on 03/07/2024).

[8] Eric Dubois et al. "A Systematic Approach to Define the Domain of Information System Security Risk Management". In: *Intentional Perspectives on Information Systems Engineering* (May 2010). ISSN: 978-3-642-12543-0. DOI: 10.1007/978-3-642-12544-7_16.

[9] *Filesystem in Userspace*. en. Page Version ID: 1198800987. Jan. 2024. URL: https://en.wikipedia.org/w/index.php?title=Filesystem_in_Userspace&oldid=1198800987 (visited on 02/12/2024).

[10] freedesktop. *systemd.exec*. URL: https://www.freedesktop.org/software/systemd/man/latest/systemd.exec.html (visited on 05/14/2024).

[11] *FUSE — The Linux Kernel documentation*. URL: https://www.kernel.org/doc/html/next/filesystems/fuse.html (visited on 02/12/2024).

[12] Daniel Ganji et al. “Approaches to Develop and Implement ISO/IEC 27001 Standard - Information Security Management Systems: A Systematic Literature Review”. In: 12 (Dec. 2019), pp. 228–238.

[13] Google. *General-purpose machine family for Compute Engine | Compute Engine Documentation*. en. URL: https://cloud.google.com/compute/docs/general-purpose-machines (visited on 04/27/2024).

[14] Grant Gross. *White House urges developers to dump C and C++*. en. Feb. 2024. URL: https://www.infoworld.com/article/3713203/white-house-urges-developers-to-dump-c-and-c.html (visited on 05/14/2024).

[15] Matthew Hoekstra et al. “Using innovative instructions to create trustworthy software solutions”. In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP ’13. New York, NY, USA: Association for Computing Machinery, June 2013, p. 1. ISBN: 978-1-4503-2118-1. DOI: 10.1145/2487726.2488370. URL: https://doi.org/10.1145/2487726.2488370 (visited on 12/29/2023).

[16] Intel. *Architecture Specification: Intel® Trust Domain Extensions (Intel® TDX) Module*. Feb. 2023. URL: https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html (visited on 04/08/2024).

[17] Intel. *Code Sample: Intel® Software Guard Extensions Remote Attestation...* en. URL: https://www.intel.com/content/www/us/en/developer/articles/code-sample/software-guard-extensions-remote-attestation-end-to-end-example.html (visited on 01/21/2024).

[18] Intel. *Intel® Software Guard Extensions (Intel® SGX)*. en. URL: https://download.01.org/intel-sgx/sgx-linux/2.23/docs/Intel_SGX_Developer_Guide.pdf (visited on 03/07/2024).

[19] Intel. *Intel® TDX Connect Architecture Specification*. Mar. 2023. URL: https://cdrdv2-public.intel.com/773614/intel-tdx-connect-architecture-specification.pdf (visited on 03/07/2024).

[20] Intel. *Intel® TDX Virtual Firmware Design Guide*. Dec. 2023. URL: https://cdrdv2-public.intel.com/733585/tdx-virtual-firmware-design-guide-rev-004-20231206.pdf (visited on 03/07/2024).

[21] Intel. *Intel® Trust Domain Extensions (Intel® TDX)*. Feb. 2023. URL: https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html (visited on 04/08/2024).

[22] Intel. *Performance Considerations: Intel® Trust Domain Extensions*. en. Sept. 2023. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/trust-domain-extensions-on-4th-gen-xeon-processors.html (visited on 05/15/2024).

[23] ISO. *ISO/IEC/IEEE 42010:2022(en), Software, systems and enterprise — Architecture description*. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-2:v1:en (visited on 05/15/2024).

[24] *jgraph/drawio*. original-date: 2016-09-06T12:59:15Z. May 2024. URL: https://github.com/jgraph/drawio (visited on 05/15/2024).

[25] Hugo Krawczyk. "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols". en. In: *Advances in Cryptology - CRYPTO 2003*. Ed. by Gerhard Goos et al. Vol. 2729. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 400–425. ISBN: 978-3-540-40674-7 978-3-540-45146-4. DOI: 10.1007/978-3-540-45146-4_24. URL: http://link.springer.com/10.1007/978-3-540-45146-4_24 (visited on 02/01/2024).

[26] Philippe Kruchten. "Architectural Blueprints—The "4+1" View Model of Software Architecture". en. In: ().

[27] *libfuse/libfuse*. original-date: 2015-12-19T20:27:34Z. Feb. 2024. URL: https://github.com/libfuse/libfuse (visited on 02/12/2024).

[28] libsodium. *Internals*. en. URL: https://doc.libsodium.org/ (visited on 01/21/2024).

[29] OMG. *The OMG® Specifications Catalog*. URL: https://www.omg.org/spec/ (visited on 05/15/2024).

[30] *Pricing | Compute Engine: Virtual Machines (VMs)*. en. URL: https://cloud.google.com/confidential-computing/confidential-vm/pricing (visited on 05/14/2024).

[31] Nick Rozanski and Ein Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. 2nd ed. Addison-Wesley Professional, Oct. 2011. ISBN: 978-0-321-71833-4.

[32] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not". In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. Aug. 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357. URL: https://ieeexplore.ieee.org/document/7345265 (visited on 12/16/2023).

[33] *seccomp - manned.org*. URL: https://manned.org/seccomp.2 (visited on 05/14/2024).

[34] Anderson Luiz Silverio, Marcelo Carlomagno Carlos, and Ricardo Felipe Custodio. "Efficient Data Integrity Checking for Untrusted Database Systems". en. In: (2014).

[35] The Mitre Corporation. *CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC™)*. URL: https://capec.mitre.org/ (visited on 05/13/2024).

[36] The United Nations. *THE UNITED NATIONS GUIDE ON PRIVACY-ENHANCING TECHNOLOGIES FOR OFFICIAL STATISTICS*. 2023. URL: https://unstats.un.org/bigdata/task-teams/privacy/guide/2023_UN%20PET%20Guide.pdf (visited on 04/08/2024).

[37] *Top Data Science Courses - Learn Data Science Online*. en. URL: https://www.coursera.org/search?query=data%20science&topic=Data%20Science (visited on 05/15/2024).

Appendix

A FUSE - Filesystem in Userspace

FUSE is a software interface for creating filesystems managed by userspace handlers. This means that file operations, regularly handled by the OS kernel, can have custom implementation defined by the user. It is demonstrated in Fig. 15, where response to the Unix 'ls' command is provided by a user defined './hello' method. FUSE is especially useful for extending the abstraction that the file system interface provides. For example, when the underlying files actually exist in cloud, or the data written or read from files needs to be encrypted/decrypted, the users of the file system do not have to know that.

FUSE composes of kernel driver (FUSE in Fig. 15), libfuse library and user defined handlers. The first two components are provided by the FUSE project [27, 11]. The last one is up to Sharemind HI Developers to develop.

The kernel module is responsible for mounting the filesystem and forwarding request to handlers. The libfuse library assists developers of custom handlers in communication with the kernel module.

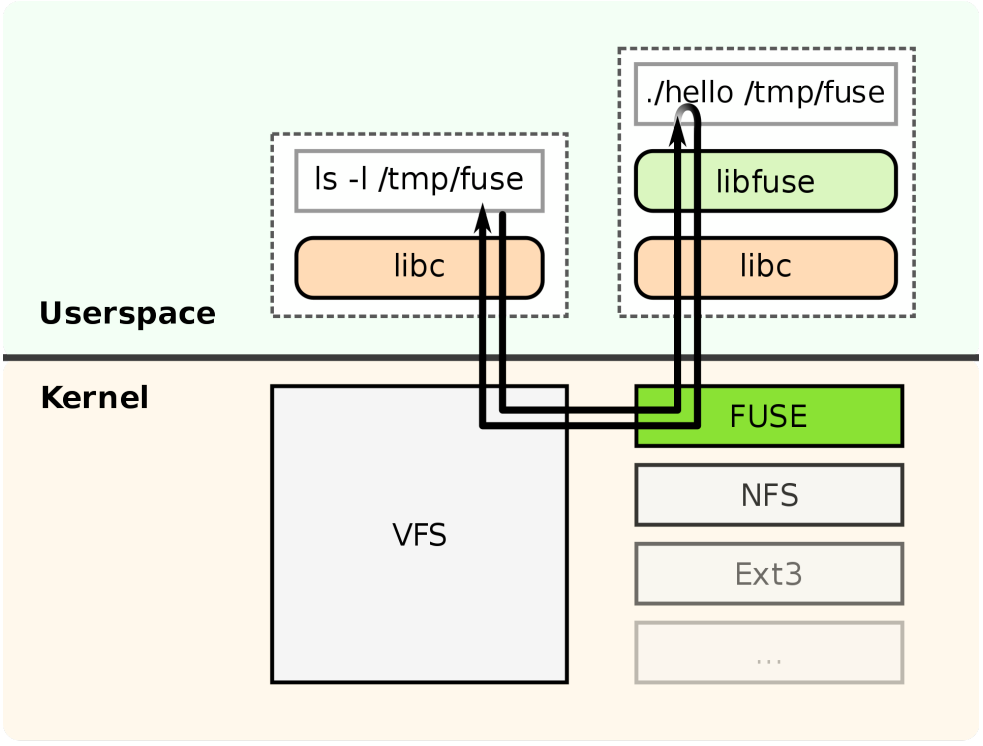

Figure 15. Overview of FUSE as depicted in Wikipedia[9]

B BPMNs with Risks

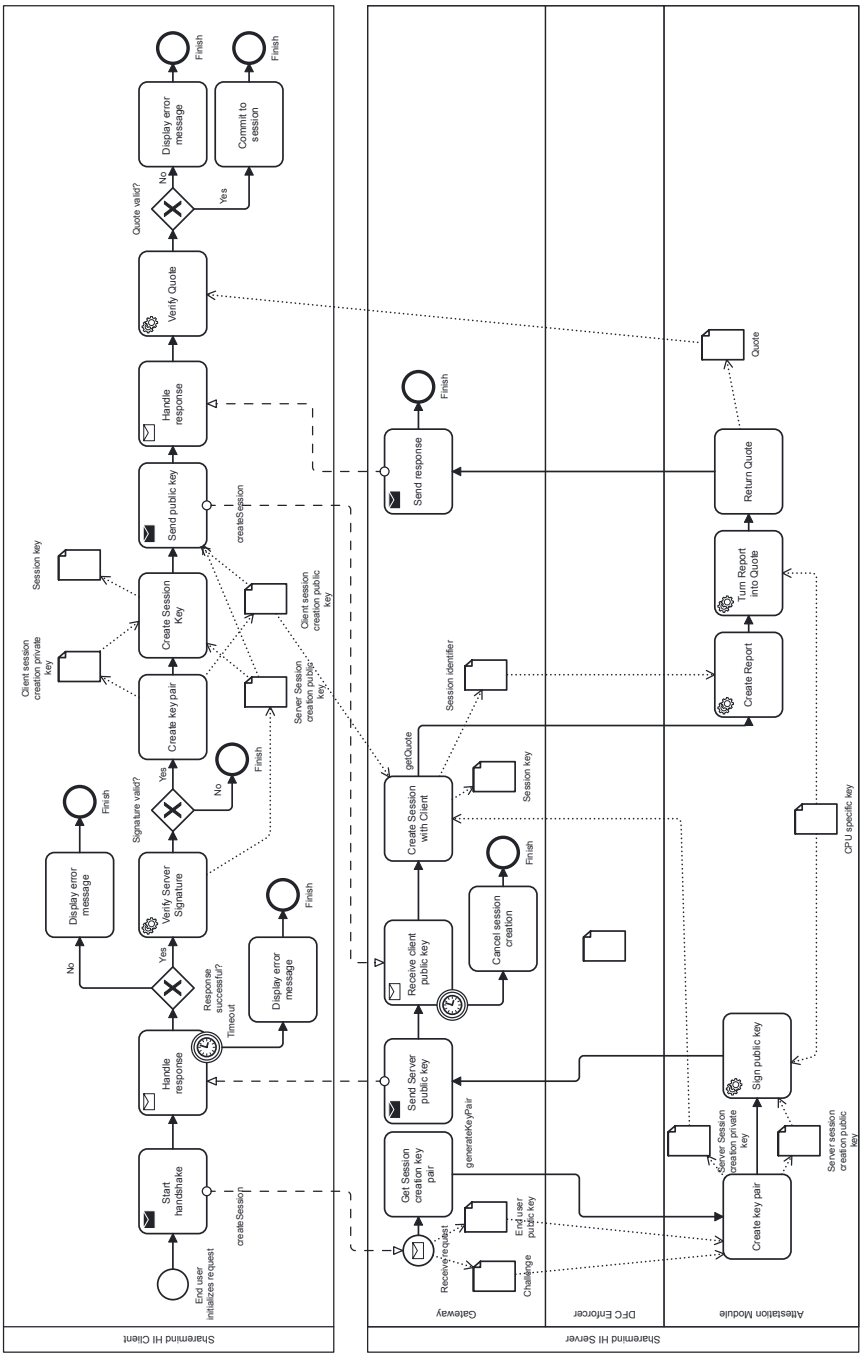

Figure 16. Session creation diagram BPMN diagram with TDX-specific risks marked in red

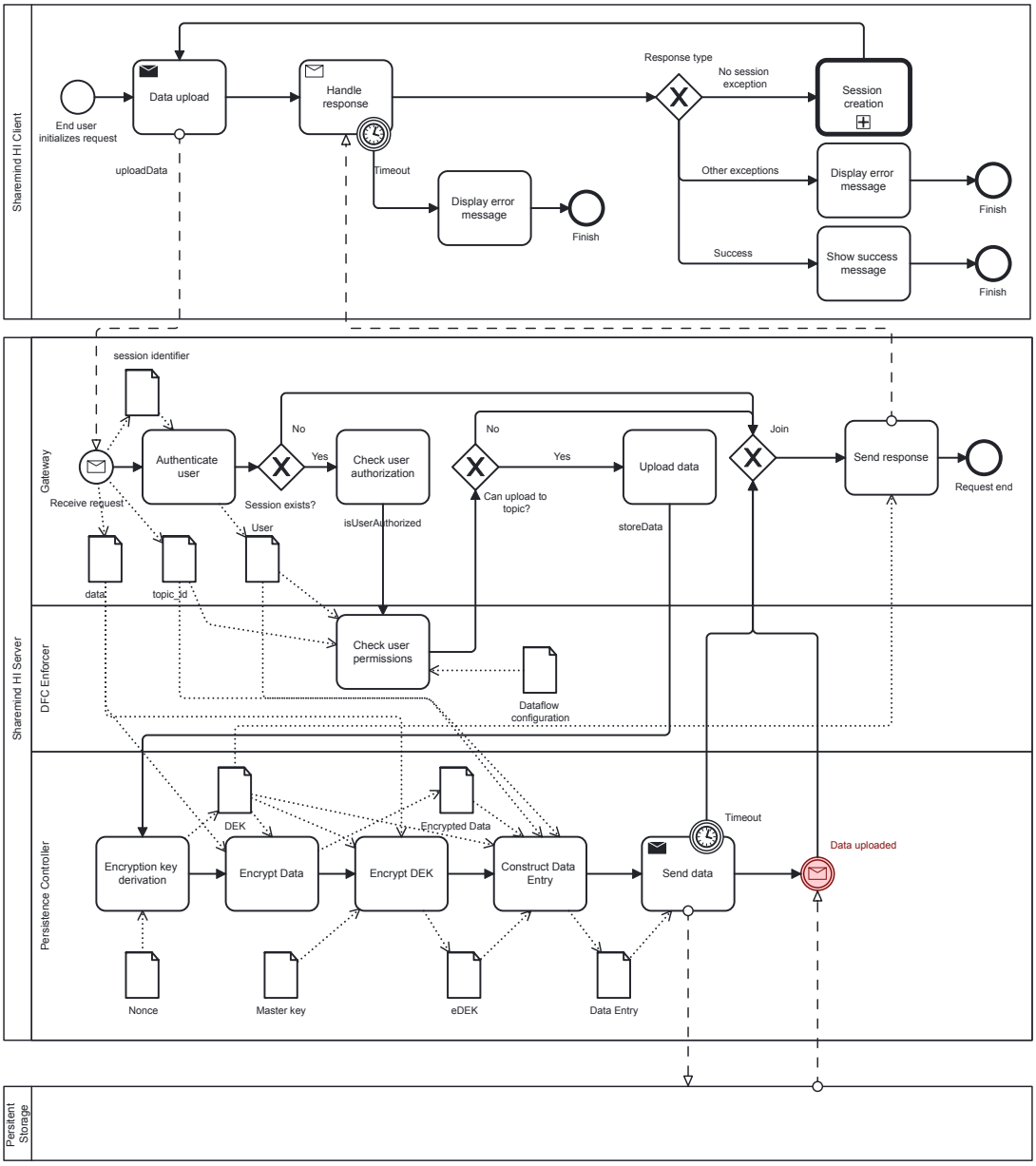

Figure 17. Data upload process BPMN diagram with TDX-specific risks marked in red

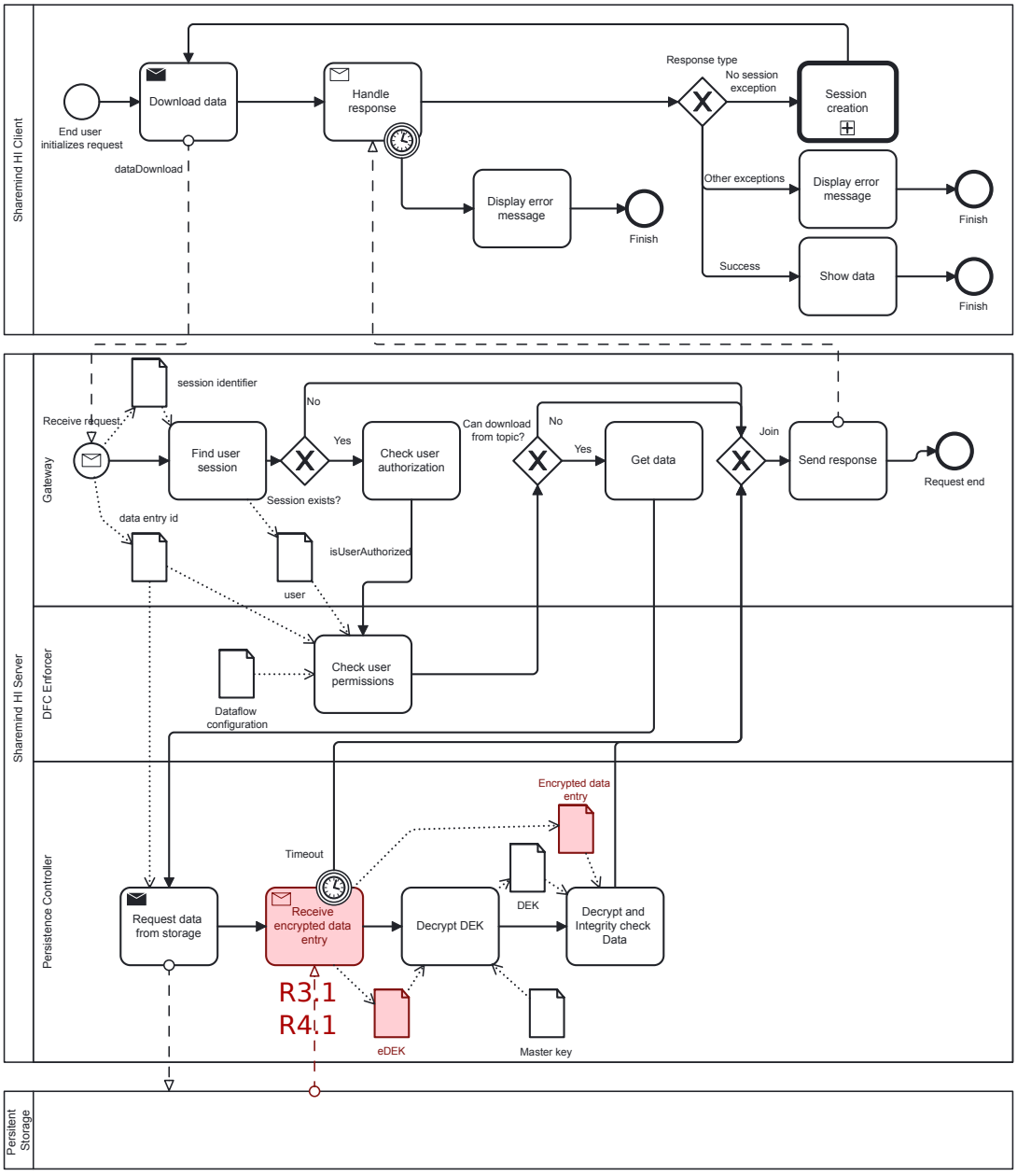

Figure 18. Data download process BPMN diagram with TDX-specific risks marked in red

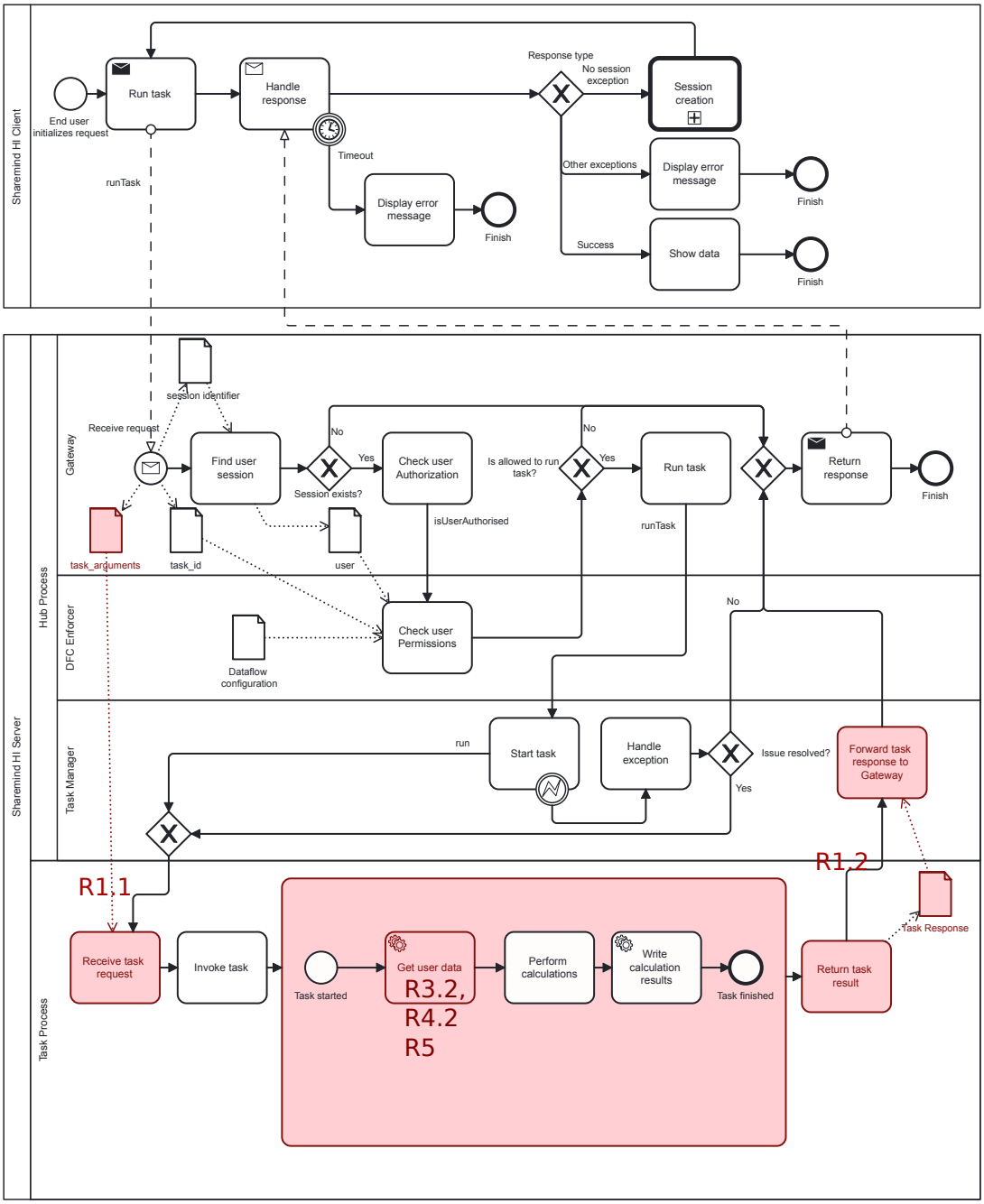

Figure 19. Run task process BPMN diagram with TDX-specific risks marked in red

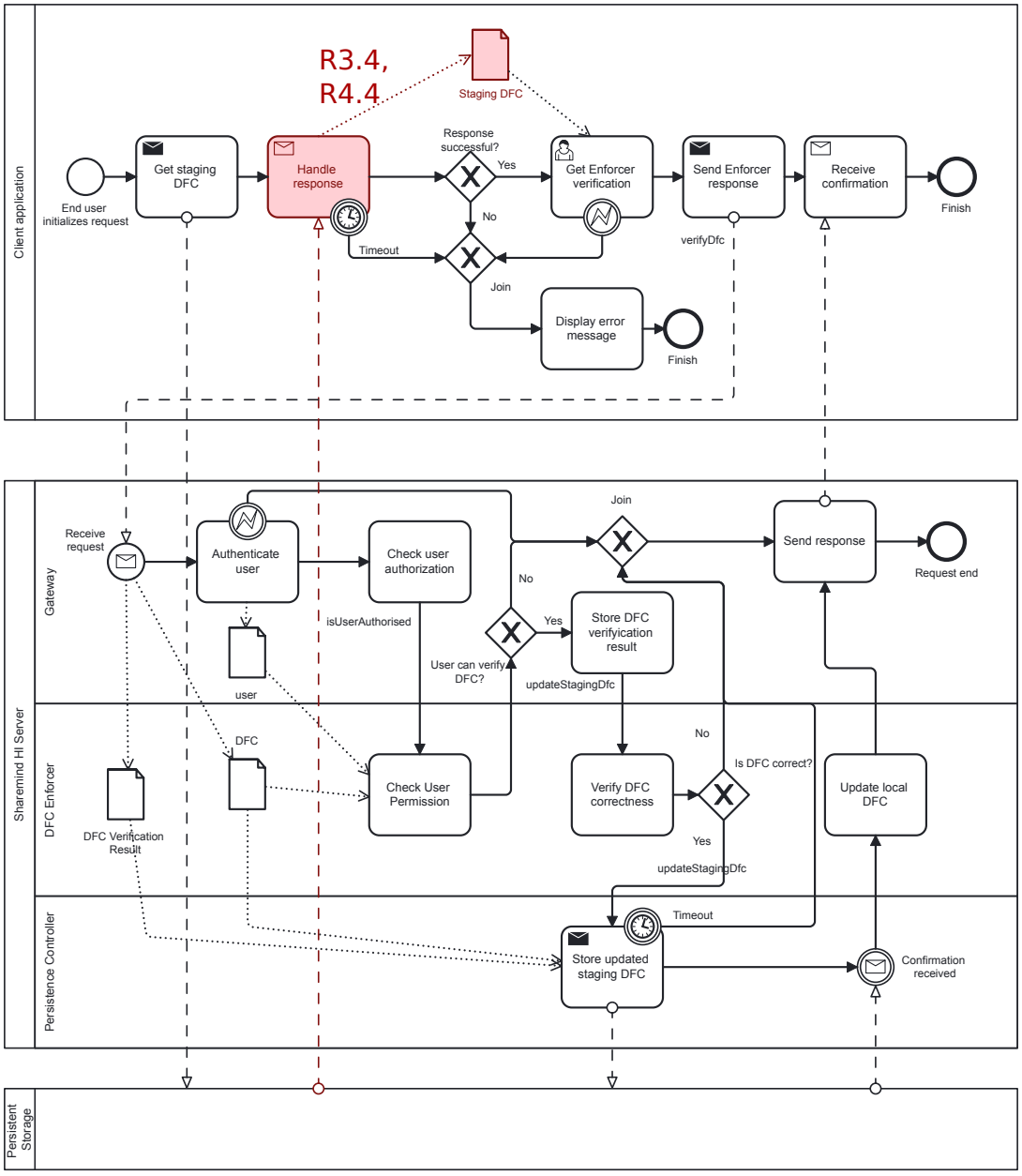

Figure 20. DFC verify process BPMN diagram with TDX-specific risks marked in red

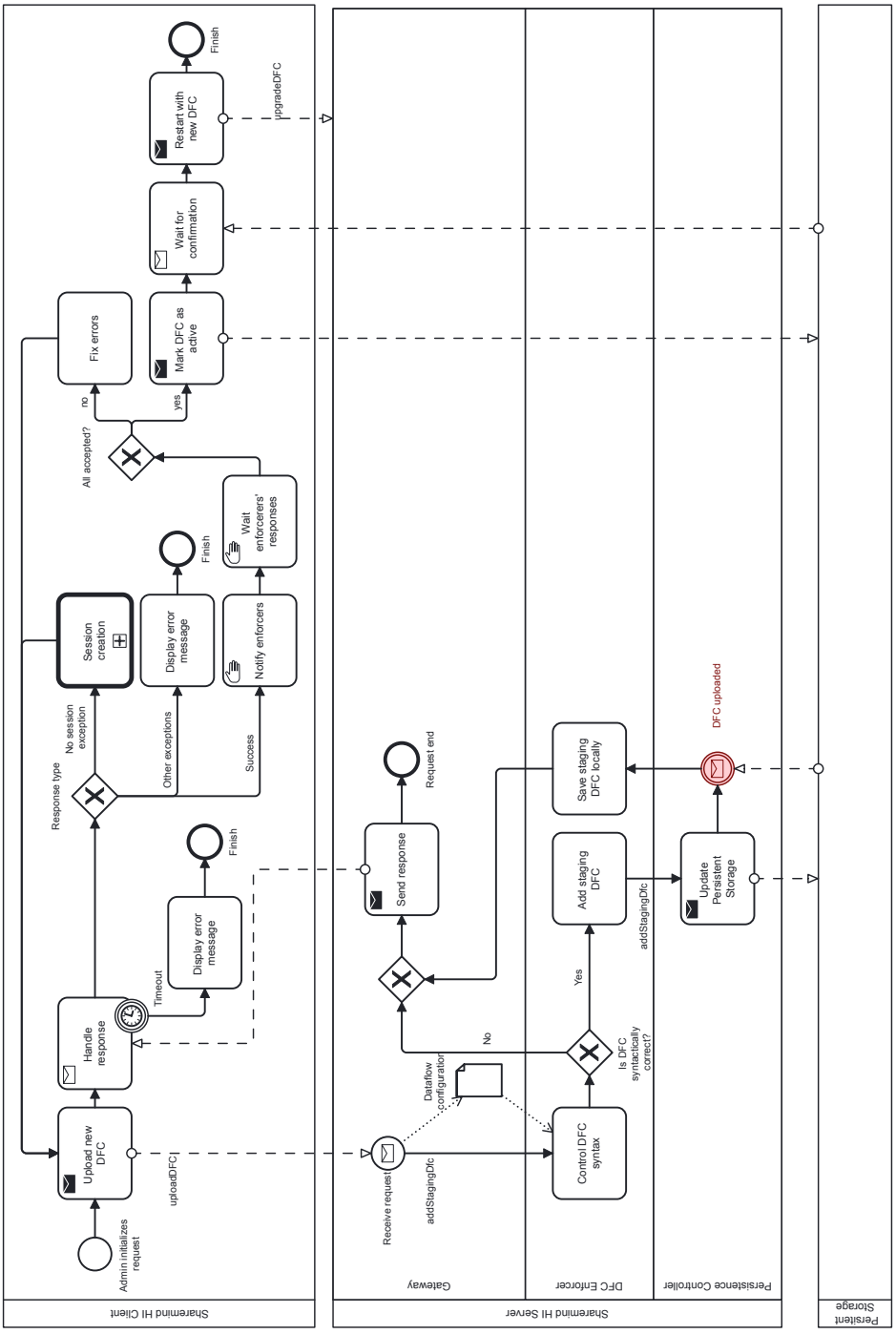

Figure 21. DFC upgrade process BPMN diagram with TDX-specific risks marked in red

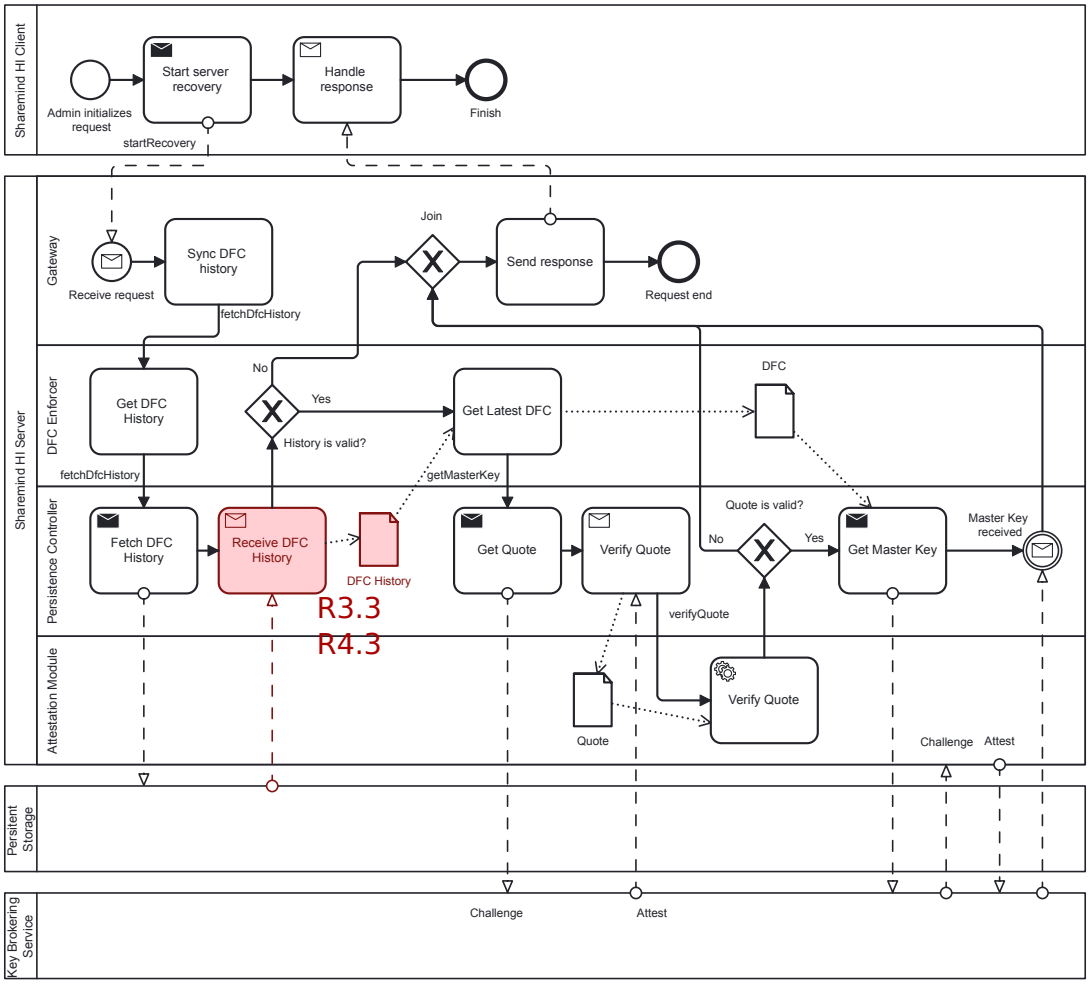

Figure 22. Server recovery process BPMN diagram with TDX-specific risks marked in red

C Performance Test Code

The test code and analysis code is available in Github through following link `https://github.com/Ratsemaat/thesis-perf-tests`. Furthermore for convenience, the test code is duplicated in the next two subsections.

C.1 PT1

```
int main(int argc, char** argv) {
  grpc::ChannelArguments channel_args;
  channel_args.SetMaxSendMessageSize(1024 * 1024 * 1024); // 1 GB
  channel_args.SetMaxReceiveMessageSize(1024 * 1024 * 1024); // 1 GB

  char* host_env = getenv("HOST");
  std::string host_address = host_env ? host_env : "localhost";
  std::string server_address = host_address + ":50051";

  HIMockerClient greeter(grpc::CreateCustomChannel(
     server_address, grpc::InsecureChannelCredentials(), channel_args
        ));

  greeter.InitSession();

  std::ofstream out;
  for (size_t i = 0; i < 8; i++)
  {
    out.open("/tmp/" + std::to_string(i) + ".txt");
    for (size_t j = 0; j < 3000; j++){
      auto str = random_string(pow(10, i));
      auto start = std::chrono::high_resolution_clock::now();
      greeter.SendData(str);
      auto stop = std::chrono::high_resolution_clock::now();
      auto duration = std::chrono::duration_cast<std::chrono::
         microseconds>(stop - start);
      out << duration.count()<< "us;" <<std::endl;
    }
    out.close();
  }
  return 0;
}
```

Listing 1. Client code

```
Status SendData(grpc::ServerContext* context, const SendDataRequest*
    request, Empty* response) {
   try {
     auto start = std::chrono::high_resolution_clock::now();
         CURL* curl = curl_easy_init();
     if (!curl) {
         std::cerr << "Failed to initialize cURL" << std::endl;
         return grpc::Status(grpc::StatusCode::UNKNOWN, "Failed to
             initialize cUR");}


     std::string url = "localhost:8081";
     curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
     curl_easy_setopt(curl, CURLOPT_POST, 1L);
     curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE, (request ->
         message()).length());
     curl_easy_setopt(curl, CURLOPT_POSTFIELDS, request->message().
         c_str());
     std::string response_data;

     // Send the HTTP GET request
     CURLcode res = curl_easy_perform(curl);
     if (res != CURLE_OK) {
         std::cerr << "Failed to perform cURL request: " <<
             curl_easy_strerror(res) << std::endl;
         curl_easy_cleanup(curl);
         return grpc::Status(grpc::StatusCode::UNKNOWN, "Failed to
             perform cURL request");}

     curl_easy_cleanup(curl);
     auto end = std::chrono::high_resolution_clock::now();
     auto elapsed = std::chrono::duration_cast<std::chrono::
         microseconds>(end - start);
     std::cout <<"size"<<(request->message()).length()<<";time"<<
         elapsed.count() << std::endl;

     return grpc::Status::OK;
   } catch (std::exception e) {
       std::cout << e.what()<< std::endl;
       return grpc::Status(grpc::StatusCode::UNKNOWN, "");
   }
 }
```

Listing 2. Server side implementation of SendData

```
svr.Post("/", [&](const auto& req, auto& res, const ContentReader &
    content_reader){
  auto start = std::chrono::high_resolution_clock::now();

  if (req.is_multipart_form_data()) {
    MultipartFormDataItems files;
    content_reader(
      [&](const MultipartFormData &file) {
        files.push_back(file);
        return true;
      },
      [&](const char *data, size_t data_length) {
        files.back().content.append(data, data_length);
        return true;
      });
  } else {
    std::string body;
    content_reader([&](const char *data, size_t data_length) {
      body.append(data, data_length);
      return true;
    });
  }
  res.status = 200;

  auto end = std::chrono::high_resolution_clock::now();
  auto len = std::chrono::duration_cast<std::chrono::microseconds>(
     end - start);
  std::cout<<"Time in task:"<<len.count()<<std::endl;
});
```

Listing 3. Task implementation of task run endpoint

C.2 PT2

```
using namespace std::chrono;

size_t sizes[] = {1,10,100,1000,10000,100000, 1000000, 10000000, 100000000, 500000000};
size_t sizes_sizes[] = {1000,1000,1000,1000,1000,1000,1000,100,10,1};

int make_a_query(pqxx::connection & C, size_t size, int numer){
     try {
      if (C.is_open()) {
         pqxx::work w(C);
         pqxx::result r = w.exec("SELECT * from indexed_data_"+ std::to_string( size )
             + "_byte WHERE id="+std::to_string(numer));
         return 1;
      } else {
         std::cout << "Can't open database" << std::endl;
      }
      return 0;
  } catch (const std::exception &e) {
      std::cerr << e.what() << std::endl;
      return -1;
   }
}

int main (int argc, char* argv[] ) {
    try{
    std::ostringstream oss;
    //Initialize connections
    oss << "dbname = postgres user=postgres password=postgres host=" << getenv("DB_HOST
        ") << " port=5432" ;
    pqxx::connection C(oss.str());
        std::ofstream myfile;
        myfile.open ("t1_results.txt");
        make_a_query(C, sizes[0], 1);
        for (size_t i = 0; i < 10; i++)
        {
            auto start = high_resolution_clock::now();
            for (size_t j = 1; j <= sizes_sizes[i]; j++){
                if (make_a_query(C, sizes[i], j) < 1){
                    throw std::runtime_error("");
                }
            }
            // Get ending timepoint
            auto stop = high_resolution_clock::now();
            auto duration = duration_cast<milliseconds>(stop - start);
            myfile<<sizes[i]<<";"<< duration.count()<< "ms\n";
            std::cout <<sizes[i]<<";"<< duration.count()<< "ms" << std::endl;
        }
        myfile.close();
        return 0;
    }
    catch (const std::exception &e) {
      std::cerr << e.what() << std::endl;
      return -1;
   }
}
```

Listing 4. PT2 server-side code

D Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Herman Rull**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Towards practical privacy-preserving data analysis with Intel TDX-based Sharemind HI,
   *(*title of thesis)

   supervised by Armin Daniel Kisand and Raimundas Matulevičius.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Herman Rull
17.05.2024