Secure Testing-as-a-Service

A Shielded Execution Approach

MSc in Cloud Computing
Cork Institute of Technology

Research Project by: Jonathan Zinger
Supervised by: Eoin O'Regan

Abstract

Cloud computing provides customers with the ability to reduce cost in regards execution of computation tasks and resource utilization. Using cloud solutions allows customers to quickly and simply provision various resources which can be used to store data, provision whole platforms, run applications and so on without the need to maintain a physical infrastructure.

Testing-as-a-service (TaaS) is a cloud solution which provides customers with all of the advantages associated with cloud solutions and with additional testing capabilities.

However the advantages associated with cloud computing are often outweighed by the risks which are inherent to cloud solutions: by utilizing the cloud service provider's resources, customers trust the cloud service provider with full access to their data. TaaS solutions also suffer from the same risks and the leak of customer data can cause significant damage the customer.

This work presents a novel design for an end to end TaaS solution deployed in untrusted cloud environments. We utilize the security capabilities which are provided by Intel's SGX and dynamically extend the trust from it to the entire solution. Providing strong security guarantees towards confidentiality and integrity of customer data.

Our solution protects customer data at all levels: at rest, in transit and while active under a strict threat model. We show that our design can be implemented using commodity hardware and commodity software with minimal modification.

Introduction

Cloud computing is changing the way customers handle computation tasks execution and resource utilization. They are now able to quickly and simply provision what is, in essence, unlimited resources which can be used to store data, provision whole platforms, run applications and so on [1]. Cloud service offerings include:

- Virtualized hardware, allowing customers to provision the hardware they need.
- Virtually unlimited storage (such as what is available through Amazon's S3) allowing customers to store the data they want.
- Software platforms (such as operating systems, services…) which allow customers to take a thin client approach to accessing the infrastructure.

These services allow companies to cut down on costs by provision the resources that they need when they need it with increased flexibility without needing to maintain them [2].

Software testing is one of the main phases of the software engineering life cycle [2]. It is a time and resource consuming task. It is important to understand that the more complex the application and functionality being tested the more resources will be consumed while testing it.

In the traditional, in house datacenter approach, resources are not always available as they might already be provisioned or simply didn't exist. This would cause QE personnel to be idle while waiting for new hardware to become available or execute tests on limited resources.

It would make sense for customers to migrate their software testing infrastructure to the cloud. A cloud solution would provide scalability, easy resource provisioning and simple test environment setup, allowing QE personnel to simplify the testing process. [2].

Customers can utilize cloud services even more. There are a number of services which offer Testing-as-a-Service (TaaS), which offer different functionalities: functional testing [3] [4], static code analysis [5], web services tests [6], mobile tests [7], etc. These services offer automatic provisioning of resources in accordance to the tests needs, adjusting the load on application as requested, pre-defined test cases for known standards etc.

However, one of the issues plaguing cloud solution which prevents a wider cloud solution adoption is security. If we put aside cloud services' general security issues, the issues which are unique for TaaS solutions are [8] [9] [10]:

- Test case leak
- Test result leak
- Test data leak
- Application binaries leak

Hence it is important that a TaaS solution provide customers with data protection. In this work we use new technology in order to design a secure TaaS solution which will provide customers with the functional benefits of a TaaS, while protecting their data.

Our research will be divided in to 5 sections:

1) ***Cloud Computing*:** Presents the basic concepts and models upon which TaaS is based.
2) ***TaaS:*** Presents a definition of TaaS solution, TaaS services, solution architecture, benefits derived from it and current implementation.
3) ***Security Concerns*:** Presents security concerns which relate to SaaS and TaaS solutions and current approaches towards handling these concerns.
4) ***Solution Design*:** Proposes a solution to the security concerns raised in section 3 by using the software development lifecycle approach.
5) ***Discussion*:** Discusses the solution proposed in section 4, proves its correctness and discusses limitations and future work.

At the end of this paper we will detail the conclusion of this research. We will summarize why TaaS is important, what are the risks and security concerns inherent in it and why our solution is best fit to handle the security concerns we raised.

The novelty of this work is derived from our threat model. By the end of this work we will show the no other TaaS solutions or approaches provide protection under the defined threat model while providing the necessary functionality.

1 Cloud computing

In order to understand the security vulnerabilities inherent to the TaaS approach, we must first understand what it is. As such we seek to understand its underlining motivation, concepts and architecture. In this section we will explain what are the basic concepts and models upon which TaaS is based.

1.1 Cloud computing

According the National Institute of Standards and Technology (NIST) [1] cloud computing is:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models and four deployment models."

1.2 Characteristics

This section details what are the essential characteristics of a cloud solution [1]:

- ***On demand self-service*:** Allows customers to provision resources and capabilities as needed, without the need to interact with a cloud service provider's (CSP) sales or service representative. The cloud solution will provision the resources automatically.
- ***Broad network access*:** The CSP's resources are available over the network and accessed through a standard mechanism exposed via user interfaces.
- ***Rapid elasticity*:** Allows customers to elastically provision and release resources, allowing them to easily scale resource utilization according to their needs. In essence customers view resources as virtually limitless and constantly available at any capacity.
- ***Measured service*:** The CSP controls and optimizes resources utilization by measuring the service resource consumption. Resource utilization is monitored, controlled and reported allowing the CSP and the customer to understand how resources are consumed.

- **Resource pooling:** The CSP's resources are pooled together allowing multiple customers to be serviced via a multi-tenant model. Physical and virtual resources are assigned dynamically according to customers' needs. This guarantees that clients are not aware off the resources utilized by the application, providing a high level of abstraction.

1.3 Service models

This section details the different service models which are provided by cloud solutions [1]:

- ***Software-as-a-Service (SaaS)*:** Provides customers with access to applications running on infrastructure which is managed and controlled by the CSP. The customer can access applications through various user interfaces.
- ***Platform-as-a-Service (PaaS)*:** Provides customers with the ability to deploy applications (consumer applications, libraries, services, etc.) onto the CSP's cloud infrastructure. The CSP controls and manages the underlining resources and the customer controls the deployed applications.
- ***Infrastructure-as-a-Service (IaaS)*:** Provides customers with the ability to provision resources (processing, storage, network, etc.) which are used to run applications. The CSP controls the infrastructure and the customer controls OS, storage and deployed applications.

1.4 Deployment models

This section details the different ways in which a cloud solution can be deployed [1]:

- ***Private cloud*:** The CSP's infrastructure is provisioned for the use of one customer (usually an enterprise) composed of several tenants. The infrastructure is managed by the customer or by a 3rd party (enterprise, academic institute, government agency, etc.).
- ***Public cloud*:** The CSP's infrastructure is provisioned for the use of the general public. It is managed by a 3rd party and exists within the party's premise.
- ***Community cloud*:** The CSP's infrastructure is used and managed of a community of customers (such as different organizations) who have shared concerns.
- ***Hybrid cloud*:** The CSP's infrastructure is composed of the above deployment models bound by standardized technology which enables data and application portability.

2 Testing-as-a-Service (TaaS)

This section will detail what constitutes as a TaaS solution, what services should be provided by a TaaS solution. A generic TaaS solution architecture will be displayed, the benefits of using a TaaS solution will be discussed and several TaaS solutions will be detailed.

2.1 TaaS solution definition

TaaS solutions focus on the following application deployment scenarios [2]:

- ***TaaS for cloud deployed web applications*:** Focuses on the validation of cloud deployed web applications using the solution's testing services. This approach utilizes elastic computing resources provided by the CSP infrastructure to run large-scale test simulations.
- ***TaaS on clouds*:** Focuses on validation of cloud applications and SaaS solutions deployed and executed on cloud infrastructure or cloud platform. This approach focuses on validation of application scalability, multi-tenancy, connectivity protocols and the API's provided by the SaaS solution being tested.
- ***TaaS over clouds*:** Focuses on validation of SaaS applications across different clouds models (private cloud, public cloud, etc.). When faced with a hybrid cloud infrastructure, diverse on-demand testing services are provided and delivered by the solution.

2.2 Services provided by a TaaS solution

TaaS solutions provide customers with testing services which focus on different application deployment scenarios. The following functionality should be provided by a TaaS solution [2]:

- ***Multi-tenant test modeling*:** As stated in section 1.2 CSP resources are pooled together using a multi-tenant model, allowing tests to execute only on their provisioned resources.
- ***Scalable test environment service*:** As stated in section 1.2 using a cloud solution allows for simple resource scaling in accordance to customer needs. This allows customers to customize their testing environments by provisioning desired resources.
- ***Test management service*:** Provides customers with the ability to manage test suites. A test suite is comprised of test cases, test data, test results and any other necessary resources such as scripts and metadata. Managing test suites includes: creation, deletion and update.

- ***Test on demand execution and control service*:** Provides customers with the ability to execute on-demand tests and schedule test executions according to their needs.
- ***Test solution integration and composition service*:** Provides customers with the ability to execute pre-defined test suites. These test suites are based on different test models, algorithms, scenarios, technologies and certificates.
- ***Test tracking and monitoring service*:** Provides customers with the ability to monitor the progress and performance of executed test cases allowing them to fine tune their tests.
- ***Test environment simulation service***: Provides customers with the ability to simulate various scenarios allowing for the testing of different aspects of the application. This allows customers to simulated workloads, GUI in different environments, load test connectivity, etc.
- ***Contracting and billing service*:** Provides customers with billing and account management capabilities. A cloud solution needs to provide their customers with these capabilities [11], allowing them to provision testing services in accordance with a pre-defined pricing model.

2.3 TaaS architecture

In order to provide the services presented in section 2.2 and to better understand what are the security concerns faced by TaaS solutions we must specify the TaaS solution architecture. A generic TaaS solution architecture, deployment model and behavior model are described in [12].

The solution is deployed as public cloud using Xen VMs on physical hosts with customer facing user interfaces. The TaaS architecture is compromised of the following layers:

Test service tenant and test service contributor layer

Provides various user interfaces providing users with access to the solutions functionality. The following sublayers exist:

- ***Test service tenant sublayer*:** Customers interact with testing services provided by the solution via Web application, IDE plugin or public API.
- ***Test service contributor sublayer*:** Contributors can publish and deploy their own custom testing service and VM images to the solution.

Test case management layer

This layer acts as a testing service bus and a bridge between tenants and the testing services. Responsible for clustering test cases according to capabilities, dispatching test cases to VMs, collecting monitored data and publishing testing services and VM images.

Test resources management layer

This layer provides the TaaS solution with the resource needed for its operation. Provides cloud services (billing, user management, etc.), monitors hosts, creates and monitors VMs and allocates testing resources to test cases. This layer is comprised of the following sublayers:

- ***Preprocessor:*** Represents operation performed before test case execution, such as: clustering test cases according to needs, high level scheduling decisions, assigning hosts to test cases.
- ***Resource management:*** Represents capabilities used for resource management, such as: load balancing, VM management, physical host deployment. It provides various cloud services.
- ***Computing node:*** Represent VMs who are deployed by the VM controller to be used for test case execution.

Test case layer

This layer corresponds to actual executed test cases on VMs. This layer is divided in to 3 different sub layers with each layer corresponding to a specific functional area within the test execution process:

- ***Service composition layer*****:** Specifies the workflow and testing services need in order to build an execution sequence for the test case.
- ***Service pool layer*****:** Contains the testing services used for different types of tests, which are executed within a VM.
- ***Test reduce layer*****:** Collects test results from various tenants and their corresponding VMs and persists them.

Testing storage layer

This layer corresponds to the solution's storage component. Its role is to store test cases, customer application libraries and code, test service, VM images and testing results.

The article then goes on to present a diagram representing the structure and behavior of the TaaS solution. The diagram is presented in figure 1.

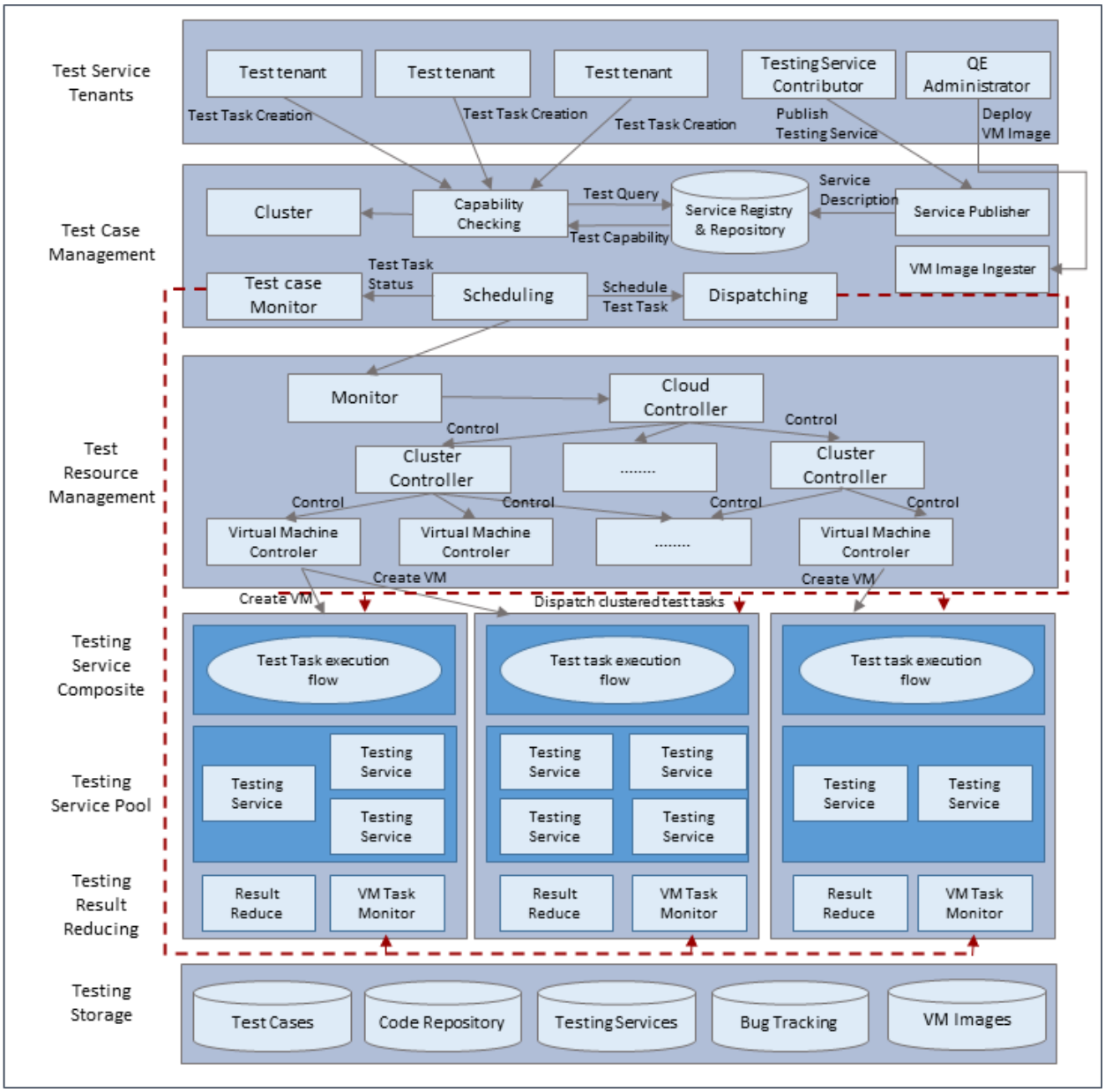

Figure 1TaaS solution architecture

2.4 Benefits of using TaaS

We have defined what TaaS is, what areas of testing it is focused on, what services it offers to its customers and what is its architecture. We will now review the benefits off adopting a TaaS solution over an in-house testing approach:

- ***Cost reduction*** [13]**:** TaaS solutions help customers reduce costs. They allow customers to provision resources and automate test execution on a pay-per-use payment model. Customers are only charged when tests have been executed. TaaS solutions manage environments, environment maintenance and setup in accordance with the defined requirements. This allows customers to save time and money on deployment, software licenses and hardware purchase and maintenance.
- ***Scalable virtual test environment*** [2]**:** TaaS solutions provide customers with varied and scalable testing environments. As stated in section 1.3 cloud solutions provide various software (SaaS) and hardware environments (IaaS). This diversity allows customers to deploy whichever environment is necessary for the execution of their testing suites.
- ***On-demand "always on" testing*****:** TaaS solutions provide customers with the ability to execute tests at any time. This allows customers to execute test suites from different campuses spread across different time zones.
- ***Multi-tenant testing*****:** TaaS solutions provide customers with the ability to execute test cases in a multi-tenant model. As stated in section 2.2 this allows different test suites to be executed on different virtualized hardware while using the same underlining infrastructure.

2.5 Current TaaS offerings

The type of TaaS solutions are as varied as the types of application offerings and the tests that can be performed on them. Describing all of them would be beyond the scope of this work. As there are various solutions available today, instead we will describe solutions which focus on up and coming technologies and different testing methodologies:

Static Testing-as-a-Service

The approach described in [5] presents the need, design and implementation of a Static TaaS solution. A Static testing solution scans customer provided code and performs syntax checking and other static code analysis in order to evaluate code correctness and discover defects. Static testing does not perform analysis on application binaries and has relatively low resource requirements.

The article provides architecture for a Static TaaS solution based on the characteristics defined. It describes the solution's logical flow, the underlining technologies used in order to implement it (such as Apache Tomcat, Hadoop and Defect Testing System) and the way customers interact with the solution.

Mobile Testing-as-a-Service

The approach described in [7] presents the need, design, implementations and challenges with regards to mobile applications testing. It states since that the rapid advance of mobile computing technology and wireless networking has caused a rise in mobile applications and SaaS solutions targeting mobile platforms and networks.

The article then goes on to present what differentiates mobile TaaS solutions from TaaS solutions, what are the unique challenges it faces, what is the testing environment infrastructure and setup required and existing approaches towards Mobile TaaS.

Architecture based service oriented architecture application testing

The approach described in [6] presents the needs for testing of service oriented architecture (SOA) applications and the difficulties of testing SOA applications. It states that SOA applications are more difficult to test; as message exchanges between participant services are hidden behind the frontend service and cannot be easily observed or controlled through the frontend service.

The article proposes a new architecture-based approach which exploits the interaction architecture of a SOA application while employing a testing architecture. This allows the solution to remove the constraints regarding the ability to observer and control the application.

3 Security concerns

This section will detail what are the security concerns which are faced by SaaS solutions, what are the security concerns which are faced by TaaS solutions specifically and detail what are the current approaches towards securing SaaS solutions.

3.1 SaaS security concerns

After reviewing the architecture and functionality of a TaaS solution in sections 2.2 and 2.3, we believe that TaaS can be classified as a subset of the SaaS service model. A TaaS solution allows customers to execute tests on infrastructure which is managed by a CSP just like the SaaS model.

In order to understand the security concerns facing TaaS solutions we most first understand what are the security concerns faced by SaaS solutions [14]:

- ***Data security*:** In the SaaS service model customer data is stored within the CSP's infrastructure, which is spread out across multiple datacenters. This is different from the on premise deployment model, in which data is stored on local infrastructure which is subject to local security and access control policies.
- ***Network security*:** In the SaaS service model sensitive data is obtained from customers, processed by the SaaS solution and stored at the CSP infrastructure. Network communication may be exposed to eavesdropping and modification by attackers.
- ***Data locality*:** In the SaaS service model customers use the services provided by the CSP to process their data. Due to the distributed nature of cloud infrastructure, customers do not know where the data is stored. If the data is stored within countries with problematic compliance and data privacy, locality of data may present an issue.
- ***Data integrity*:** In the SaaS service model applications operate under a multi-tenant model, as well as exposing application functionality via XML based APIs, SOAP and REST web services. Current standards geared towards web services data integrity are not yet mature.
- ***Data segregation*:** In the SaaS service model applications operate under the multi-tenant model. As a result data belonging to different tenants is processed and stored by the solution's services on the same physical infrastructure. Malicious tenants might try to access data which belong to other tenants.

- ***Data access*:** In the SaaS service model customers may wish to access their data from various locations. Access to data should be definable by customer in accordance to their policies.
- ***Authentication and authorization*:** In the SaaS service model applications operate on a multi-tenant model, hence management of users and users' identity is important.
- ***Data confidentiality*:** In the SaaS service model data is stored on the CSP's datacenters which might be shared across different locations and may not be owned by the CSP. When customers share data over the cloud and it leaves the CSP's control, privacy and confidentiality issues may arise.
- ***Web application security*:** In the SaaS service model customers must have access to the services provided by the solution. SaaS solutions are either deploy over the internet or behind a firewall in LAN or PC. Security vulnerabilities in the SaaS solution's web application may expose to solution to vulnerabilities. These vulnerabilities can potentially impact the deployed solution and through it the solution's entire customer base.
- ***Data breaches*:** In the SaaS service model customer data is store in the CSP's datacenters. If an attacker is able to breach that datacenter he will have access to data belonging to all customers.
- ***Vulnerability in virtualization*:** In the SaaS service model elasticity and resource pooling is achieved through virtualization. Ensuring that different VM instances running on the same physical machine are isolated from each other is an important task of virtualization. Many major virtualization vendors contain weakness that can be exploited in order to access VMs resources.
- ***Backup*:** In the SaaS service model data and services are expected to be available at all time. In order to facilitate this functionality customer data should be backed up regularly in order to allow for recovery in case of disasters.
- ***Availability*:** In the SaaS service model services are expected to be available at all times. In order to facilitate this functionality a multi-tiered architecture is adopted. The application is executing in a load-balanced farm of application instances, running on a number of servers in order to allow for scalability and high availability. Resiliency to hardware/software failures, as well as to DoS (denial of service) attacks, needs to be built into the application.

- ***Identity management and sign-on process*:** In the SaaS service model applications operate in a multi-tenant model, hence management of users and users' identity is important. Identity management (IdM) deals with identifying individuals in a system and managing resource access control. This is done by placing restrictions on the established identities. A SaaS solution should support identity management and sign on services using one of these models: Independent IdM stack, Credential synchronization or Federated IdM.

3.2 TaaS security concerns

In this section we will detail the security concerns faced by TaaS solutions. We have previously classified TaaS under the SaaS service model and as such we can apply research concerning SaaS security concerns to it. However due to the nature of services provided by TaaS solutions, we should focus on the security concerns which are inherent to them:

Test data leak

As stated in section 2.2, TaaS solutions allow customers to upload test data for their designed test suites; some test suites might even require customer or production data. The complexity or the use cases provided by test suites designers might not align with the way customers actually operate [8]. According to interviews presented in [9], industry leaders have concerns in regards to providing TaaS applications with real world data.

If customer data leaks, it would cause legal, monetary and reputation damage to the customer and to the TaaS solution. Because of these reasons that TaaS solutions must ensure that test data does not leak and if it does then it must be illegible to the attacker.

Application code leak

As stated in section 2.2 TaaS applications allow customers to create test cases which make up their test suites, as such they may need to upload application binaries or archives (such as jars) which will be used by the test cases. Some TaaS solution may also offer pre-defined test cases (such as certificate verification or code analysis) to be against application binaries and application code.

In these cases there is potential risk of that application libraries leak. While the usage of binaries mitigates this risk, but does not eliminate it as application libraries can be extracted from binaries as stated in [10]. Because of these reasons that TaaS solutions must ensure that application code does not leak and if it does then it must be illegible to the attacker.

It is important to understand that application code is a subset of test data; however the damage which can be caused by their leak is much more significant.

Test results leak

As stated in section 2.2 TaaS solutions collect test results generated by the test case execution, store them and provide customers with reports upon request. The results may contain positive information such as standard implementation; however they will undoubtedly contain test failures and application weakness such as security vulnerabilities, performance issues and various defects.

If these reports were to leak competitors can use them cause harm to the customer by discrediting and downplaying the customer application's performance, functionality and security in comparison to their own. It is because of these reasons that TaaS solutions must ensure that test results do not leak and if they do then they must be illegible to the attacker.

Test case leak

As stated in section 2.2 TaaS solutions allow customers to create test cases which are incorporated into their testing suites. These test cases are used to verify the behavior and correctness of application logic, the non-public classes and services of the application and potential application vulnerabilities and limitations.

If these test cases leak competitors can use them in order to reverse engineer the application behavior and point out potential application weaknesses and vulnerabilities and used them to discredit the application. If attackers were to get a hold of these tasks they can analyze them in order to understand the application login and weaknesses. This knowledge would allow them to

design various attacks against the application. It is because of these reasons that TaaS solutions must ensure that test cases do not leak and if they do then they must be illegible to the attacker.

3.3 Current approaches cloud service security

In this section we will present the current approaches towards securing cloud services. As stated in section 3.1, application security is a major concern facing cloud solutions. As customers look into migrating to cloud solutions the security concerns raised in section 3.2 will surely be taken into account. It is not surprising then that there are several approaches geared toward cloud service security.

3.3.1 Software assurance approach

According to [15] several approached claim that in order to confirm the security of a cloud service we must perform validation tests against a set of security requirements. We will describe several takes on this approach:

Evidence assurance approach

The approach described in [15] states that we should take a software assurance approach to demonstrate that the solution is secure. The article makes the argument that the way to demonstrate that a cloud solution is secure, is to define a set of requirements (in this case security requirements) that the cloud solution needs to fulfil and then run tests which would show if the solution fulfils these requirements.

The article later goes on to describe the merits of this approach when it comes to cloud security, a proposed implementation framework, an analysis of what requirements defining capabilities are provided to the customers and how to implement these capabilities.

Model based attack injection approach

The approached described in [16] states that threat modeling can be used in order to generate the attack scripts which correspond to the security threats which are faced by the application. It states that as each application is different from a functional architectural and technological

standpoint, threat modeling can be used to understand what type of attacks can endanger the application.

The article later goes on to describe how known attacks and known vulnerabilities are turned into attack scenarios. Attacks scripts are generated according to the testing tool that is being used. It is shown that new attacks can be created by varying available attacks scripts. Experiments using different type of attacks (DoS and Cypher suite rollback attacks) are conducted in order to show that the approach works.

3.3.2 Security-as-a-Service (SCaaS) approach

The approach described in [17] claims that CSP's should provide customer with the ability to provision which ever security capabilities they want; essentially providing Security-as-a-Service. This approach has the added value of decoupling functionality and security, moving security responsibilities to providers who "specialize" in security allowing the CSP to handle functionality [17]. We will describe some of the different takes on this approach.

SLA based protection via SPECS approach

The approach described in [18] states that customers should be able to provision and activate security capabilities through an interface. In order to provide this functionality, they define a language called SPECS which connects Service License Agreement's (SLA) security requirements and the security capabilities provided by SCaaS providers.

The SCaaS solution uses SPECS to express which security capabilities are provided and SCaaS customers use SPECS to understand which security capabilities are required. The appropriate security capabilities will then be provisioned and monitored. Monitoring is an essential to allow for SLA enforcement.

On-demand security approach

The approached described in [19] states that customers should be able to provision a VM with security capabilities which corresponds to an underlining threat model. The approach offers 7 different security levels which can be combined in order to create the desired security level.

This approach seeks to leverage secure software-hardware architecture in order in order to allow customers to select the architecture they believe best suits their needs.

The approach also proposes the use of VM migration in to upgrade/downgrade a VM's security capabilities according to customers' requests and as a reaction to an attack against the VM. If an attack is underway then the VM will be migrated to a more secure architecture which is able to deal with this type of attack.

Unified threat management based approach

The approach described in [20] states that a Unified Threat Management (UTM) solution should be used to secure cloud solutions. A UTM solution acts as a modern firewall, inspecting all communication passing through it and allowing it through reach the application only if it passes inspection.

The UTM acts as the primary interface between customers and cloud solutions and contains the security capabilities needed in order to protect cloud solutions from attacks. It is contains an advance stateful-inspection firewall with several interfaces which are used for handling various security threats [20].

3.3.3 Cryptographic cloud storage approach

According to the approach described in [21], cryptographic cloud storage is a virtual storage solution based on cryptographic techniques. It aims to gain the security advantages provided by a private cloud with the functionality and cost savings features of a public cloud.

The cryptographic storage solution strength lies in the fact that the security capabilities provided by it are achieved based on a strong cryptographic guarantee as opposed to legal, physical and access control mechanisms. The cryptographic cloud storage provides customers with the following capabilities:

- ***Data Confidentiality*:** An attacker cannot learn any information about customer. As the data is encrypted by the solution a security breach poses little to no risk for the customer.

- ***Data integrity*:** Unauthorized modification of customer data can be detected. Data is encoded for verification and the solution provides a public data integrity verification service.
- ***Data sharing*:** Customers can share their data with trusted parties. Customers can provide trusted parties with their encryption key allowing for simple data sharing.
- ***Data deletion:*** Customers can verify that data stored in the cloud us properly deleted. As the keys are stored by the customer data deletion can be achieved by erasing the decryption key.

3.3.4 Trusted computing approach

Trusted computing is an approach which seeks to resolve security issues by taking advantage of security designated hardware component and modifying software applications to take advantage of them.

The Trusted Computing Group (TCG) which has been created by several major hardware and software vendors (Intel, Microsoft, HP…) has created specifications for the protection of computer resources from threats posed by malicious external entities. The TCG has published a standard for a secure cryptoprocessor called Trusted Platform Module [22]. Trusted Platform Module (TPM) is [23]:

"A hardware cryptographic module that provides two important services, namely, secure storage and platform attestation. It consists of a processor (execution engine), volatile memory (RAM for execution) and persistent storage. It also includes cryptographic engines for random number generation, SHA-1, RSA key generation, encryption and signing. It has a special set of registers called Platform Configuration Registers (PCRs). PCRs can only be written through an extend operation, i.e., writing `X' to a PCR actually writes the SHA-1 hash of the concatenation of its current value and `X' into the PCR. Verifying the hashchain starting at the initial content of a PCR to its current value is referred to as verifying the value of a PCR."

According to pages 417-418 in [24] a TPM processor provides the following security:

- ***Secure communication*:** Allows creation of a protected path between the user and the application which is currently running. This prevents data transferred across this path from being intercepted by exterior parties.

- ***Memory curtaining*:** Provides memory protection from all external entities including the operating system and devices which can access memory directly (such as a network card). This protects data which is stored in memory during application execution from being accessed by any party except the application.
- ***Sealed storage*:** Allows data to be sealed and only be accessed when specific hardware and software system configuration are provided. This prevents data on the hard disk from being read by moving it to a different system or by creating a copy.

A host which contains a TPM processor can prevent a malicious CSP from making a copy of customer data; it would prevent external parties such as a malicious VM monitor (Hypervisor) from accessing the customer data during test case execution and it would prevent any data from being transferred to any external parties.

3.3.5 Virtual TPM approach

As we have shown in section 3.3.4 TPM is a security approach which provides applications with several critical security capabilities such as secure communication, memory curtaining and sealed storage. It relies on a trusted TMP processor which provides the root of trust for a specific host. Virtualizing the TPM would allow all virtual machine to take advantage of TPM like capabilities, increasing the security capabilities of all corresponding VMs deployed on virtualization enabled host machines.

A Virtual TPM implementation must provide the same level of security for a virtual environment as a physical TPM would provide for a physical one [25]. The implementation should focus of the following key features: protection of the virtual TPM secrets, link between the virtual TPM and VMs, extension of the chain of trust from the host machine to the VMs and key management. The following virtual TPM implementations are currently available:

Xen virtual TPM

Virtual TPM capabilities are supported by Xen 4.3 for paravirtualised VMs (VMs which are similar to the underlining physical host) [25]. Being paravirtualized the Xen virtual TPM implementation relies on the existence of physical TPM.

Xen uses a system hypervisor to create VMs which contains a software emulated TPM running on virtual TPM stub domain and an OS which is dedicated to TPM functionality. The virtual TPM stub domains are managed by virtual TPM stub manager which links the virtual TPMs to the physical TPM which is deployed on the host.

QEMU virtual TPM

Virtual TPM capabilities are supported by QEMU 1.5 [25]. Currently there are 2 different virtual TPM implementations which exist within the QEMU project. An official version based on TPM Passthrough and a non-official version which provides a fully virtual TPM. These are the offerings:

- ***TPM Passthrough***: Provides VMs with a virtual TPM which is a mapped to the physical TPM of the host. The service is designed as a backend driver for the physical TPM that communicates with an emulated TPM interface frontend.
- ***Full virtual TPM***: Provides VMs with a complete virtual TPM implementation which is not connected to a physical TPM. The service is designed as software TPM backend implementation linked with an external library called libTPMS which provides TPM emulation. The VM contains an emulated TPM interface frontend and a modified open source BIOS, which is based on SeaBIOS, to support the virtual TPM.

3.3.6 Intel Software Guard Extension approach

Intel Software Guard Extensions (Intel SGX) is a new set of processor instructions and memory access changes which are added to the Intel Architecture [26].

Applications can use SGX to execute selected code sections inside an isolated software container called an enclave. Access to data and code which are executed inside an enclave is restricted to all software entities which are executed outside the enclave. This includes system software (such host OS, hypervisor, management software, BIOS, etc.) and other enclaves. Furthermore attempts to modify an enclave's content are detected and either prevented or aborted.

SGX hardware ensures that code execution is only performed on enclave authorized locations and that unplanned exits from the enclave do not leak enclave information.

The following security capabilities are provided by SGX [27]:

1. *"Allow application developers to protect sensitive data from unauthorized access or modification by rogue software running at higher privilege levels."*
2. *"Enable applications to preserve the confidentiality and integrity of sensitive code and data without disrupting the ability of legitimate system software to schedule and manage the use of platform resources."*
3. *"Enable consumers of computing devices to retain control of their platforms and the freedom to install and uninstall applications and services as they choose."*
4. *"Enable the platform to measure an application's trusted code and produce a signed attestation, rooted in the processor, that includes this measurement and other certification that the code has been correctly initialized in a trustable environment."*
5. *"Enable the development of trusted applications using familiar tools and processes."*
6. *"Allow the performance of trusted applications to scale with the capabilities of the underlying application processor."*
7. *"Enable software vendors to deliver trusted applications and updates at their cadence, using the distribution channels of their choice."*
8. *"Enable applications to define secure regions of code and data that maintain confidentiality even when an attacker has physical control of the platform and can conduct direct attacks on memory."*

SGX can be used for true random number generation using the *RDRAND* command [28] which can then be used for generating a strong symmetric encryption key [29]. SGX also contains a public/private key pair which can be used for asymmetric encryption [30].

There is no processor support to SGX currently available. However an emulator has been shared with selected partners such as Microsoft [31] and [32].

In order to allow for data storage and communication between different enclaves and entities executing outside of the enclave the following processes are provided by SGX architecture [28]:

Sealing

Sealing is a process used by enclaves for data persistency outside the enclave's protection. While an enclave exists all data inside the enclave is protected, once the enclave is closed all data inside the enclave is destroyed.

The process of sealing is as follows: Before the enclave exists the enclave encrypts the data for confidentiality and integrity and stores it within unprotected memory. SGX does not impose any restrictions on the encryption algorithm used for encryption. Preferably a symmetric encryption algorithm should be used as an enclave can provide a symmetric encryption key, called the sealing key, via the *EGETKEY* command.

EGETKEY value is determined by one of the following policies:

1) ***Enclave identity:*** Sealing key value is set according the to the enclave's identity. In this use case the sealing key is based on the enclave's measurements. This means that only the exact same enclave can unseal the data, which prevents data migration between different enclave versions.
2) ***Sealing Identity:*** Sealing key value is set according the sealing identity. In this use case the sealing key is based on the enclave's owner measurements. This means that different versions of the same enclave can unseal the data, which allows for data migration between different versions of the enclave.

The process and details of sealing key generation are detailed in [28].

Attestation

Attestation is a process used by enclaves in order to prove to another party that the correct software is securely running inside an enclave on SGX enabled host. The SGX architecture defines two different processes towards attestation:

1) ***Local attestation*****:** Used for attestation of enclaves who are running on the same platform. The enclave produces a report using the *EREPORT* command; the report is then sent the target enclave in order to attest its identity and correctness. The target enclave verifies the attestation and sends a corresponding report the original enclave.

2) ***Remote attestation*:** Used for 3rd party components running outside the platform. In order to perform this process a dedicated enclave (quoting enclave) running on the same host is used. Local attestation is performed between the quoting enclave and the attesting enclave. The attesting enclave provides a report to the quoting enclave, which is then verified and signed using its local private key, creating a quote. These quotes are then used in order to attest the attesting enclave's identity via a public certification provided by the quoting enclave.

Both of types of attestation can be used in order to establish secure communication between the two parties, by using the enclave's public/private key pair. The details of the exact content of reports and quotes and the process used to verify them are detailed in [28].

Enclave application execution process

In order to take advantage of SGX security capabilities, an application needs to be built with SGX in mind. The process of executing software inside an enclave is different from regular software execution, as it should not contain sensitive data before execution begin. The process is explained in figure 2.

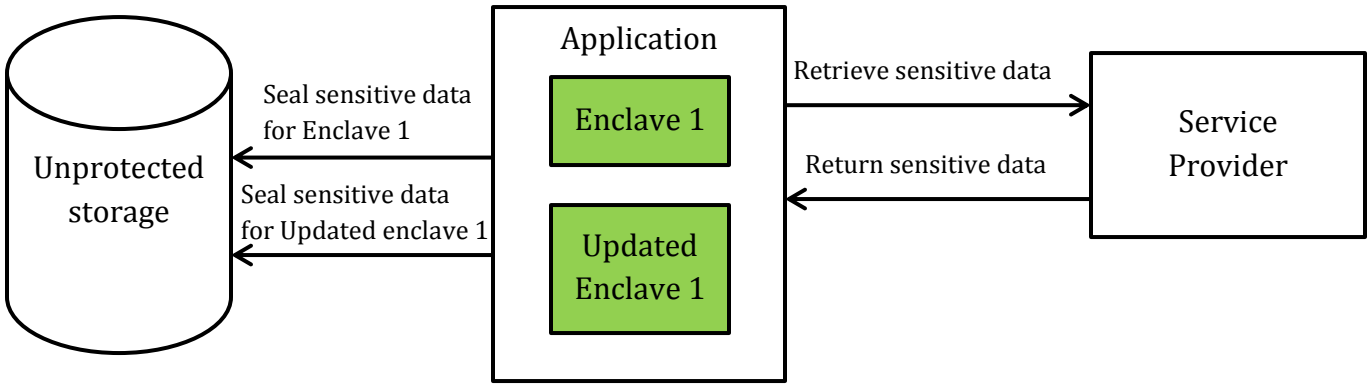

Figure 2Enclave application lifecycle

1) ***Enclave construction:*** An enclave is constructed by the untrusted application; the construction process and content of the enclave are logged and are used as the enclave's measurement.
2) ***Data request:*** A request to provision application code is sent from the enclave to the service provider. The enclave constructs a secure assertion of the hardware environment and of the enclave. It is then sent to the service provider.
3) ***Data provisioning:*** Once the service provider is assured of the enclave's trustworthiness, a secure communication channel is established and the application code and the data needed to use it are provided.
4) ***Data sealing:*** Sensitive data is encrypted by the enclave using the SGX sealing key. The data is then stored in unprotect storage.
5) ***Software update:*** When the software is updated, the older enclave's sealing key is used to unseal stored data. The new enclave's sealing key is then used seal the data again. It is then stored in unprotected storage.

4 Solution Design

In the previous sections we have reviewed some of the basic terminology around cloud solutions and their architecture. We then provided an in depth look into what constitutes as a TaaS solution, what services are provides, a TaaS solution's architecture, the advantages gained by customers adopting a TaaS solution and current TaaS solutions. We then surveyed the security concerns faced the SaaS solutions in general and TaaS solutions specifically and detailed the current approaches towards securing cloud services.

In this section we will design a TaaS solution which provides adequate protection to customer's data. We will follow the Software development lifecycle (SDLC). SDLC contains the following phases [33]: initial investigation, requirements definition, system design, implementation and testing. While different SDLC approaches differ on the number of iteration, the time spent in each iteration and so forth these phases exist throughout.

We will define our user story, threat model and requirements. We will explain why the security approaches detailed earlier do not protect the TaaS solution under the threat model we defined. We will then present our design using UML diagrams.

4.1 Initial investigation

The initial investigation phase is comprised of defining a user story and a thread model. This will allow us to derive requirements in the following phase.

4.1.1 User story

In this phase we will present a common, easy to understand scenario which will illustrate the logical workflow of a customer's interaction with our solution. We will present the stakeholders, their roles and their interaction with the system:

Corp Inc. is a software company specializing in stock trading related solutions. It is an established company which has earned the trust of several banks by providing high quality solutions with great performance. Corp Inc. is looking to reduce the costs associated with testing their applications by adopting TaaS solution.

After a long search process Corp Inc. has decided to adopt the TaaS solution proposed in this work. Corp Inc. has identifies the following software testing process stakeholders.

Adam is a QE system administrator, his responsibilities include:

- *Maintenance of testing environments.*
- *Maintenance of testing resources.*
- *Provisioning testing resources to QE personnel.*
- *Maintaining testing environment access roles.*
- *Ensuring QE personnel are assigned appropriate roles.*

Tim is a QE personnel, his responsibilities include:

- *Designing test cases.*
- *Executing test cases.*
- *Collecting test result and reporting.*

Corp Inc. would like for the migration process to the new solution to be as seamless as possible. Tim's and Adam's roles and responsibilities should remain the same, within the solution's context.

Adam's new responsibilities, as QE system administrator, include:

- *Registration of new users to the TaaS solution.*
- *Arrange billing operations.*
- *Verifying security claims.*
- *Defining what VM capabilities are necessary in order to perform testing.*
- *Defining user access roles in accordance to company policies.*
- *Ensuring QE personnel are assigned appropriate roles.*

Tim's responsibilities, as QE personnel, remain the same:

- *Designing test suites.*
- *Executing test suites.*
- *Collecting test result and reporting.*

4.1.2 Threat model

In this section we define a threat model for our solution. The threat model defines what the attacker's capabilities are and what the security assumptions we will be making.

We define the following threat model which extends to all physical hosts who are part of the CSP's infrastructure:

- We assume that any client machines located on premise are secure and can be trusted.
- We assume that the host's processor is implemented correctly and is not compromised.
- We assume that the attacker has full control of the host's hardware components (excluding the processor) including memory and all I/O devices.
- We assume that the attacker can probe the host's memory and arbitrarily alter or inject I/O including network traffic.
- We assume that the attacker controls the host's entire software stack, including the host OS, hypervisor, management software, BIOS, etc.
- We do not consider a malicious test tenant. Testing code will not attempt to leak customer data from inside the enclaves or compromise the security of our solution; however it may contain some bugs.

We don't consider the following attacks: Denial of service, network traffic analysis, hardware side-channels and fault injections. These attacks are outside the scope of our work.

It is this threat model and the assumptions made on an attacker's capabilities which differentiate our solutions for the other solutions we have shown in section 3.3.

4.2 Requirements definition

In the requirements definition phase we define user stories, project goals and other information into formally defined requirements. By observing the above user story, threat model and TaaS services in section 2.2 and TaaS security concerns in section 3.2 we can derive the following requirements:

***Functional requirements*:**

1) The customer must be able to manage test cases and test data within TaaS solution.
2) The customer must be able to perform tests using the test cases and test data provided to the TaaS solution.
3) The customer must be able to access the test results of the tests executed against the test data provided to the TaaS solution.
4) The TaaS solution provides a scalable test environment which would change according to the test suites needs.
5) The TaaS solution must be able to simulate various environments in order to test various aspects of the application.
6) The TaaS solution must provide a contracting and billing service.

***Security requirements*:**

1) The customer's test data, test results and test cases must be protected from access by any unauthorized entities.
2) The customer's test data, test results and test cases must be protected from modification by any unauthorized entities.

4.3 Weakness in current cloud security approaches

As stated in section 3.2 TaaS solutions are mainly concerned with the leak of customer data: test data, test cases and test results. If customer data leaks it would cause harm to the customer. Hence a TaaS solution needs to put the security of customers' data first.

Several approaches towards securing cloud applications are presented in section 3.3; these are divided into different approaches: security validation through software assurance, achieving security through SaaS, cryptographic data protection and hardware derived protection. Let us review these approaches discussed against to the threat model defined in section 4.1.2:

4.3.1.1 Software assurance approach

The security validation through software assurance approach states that in order to confirm the security of cloud solutions we must perform validation tests against a set of security requirements. This allows customers to verify that the tested solutions are indeed secure. We have presented a several approaches and shown how these approaches generate the test cases used for security verification.

This approach can be used to validate a cloud solution's security, however it does not prevent security issues it just detects them. The customer would still have to implement or find a TaaS solution which would then be tested; hence this approach is not applicable in our case.

4.3.1.2 SCaaS approach

The SCaaS approach allows customers to defer security to a designated service. This approach allows the CSP to separate application functionality and security, as each aspect is handled by a specialized service. Customers utilizing this type of service can decide which security capabilities they wish to enable according to their security needs. While the concept of SCaaS is interesting the applications of this concept are lacking:

SLA based protection via SPECS approach

This approach defines a language which allows customers and CSPs to define what are the security capabilities which are needed and allow for automatic deployment of these features. Since SLA's are a well-known concept for both service providers and consumers it's natural that it would be used for this purpose. It also provides customers with monitoring capabilities to make sure the SLA is met [18].

However this approach also has weaknesses, while it allows customers to request and provision the security capabilities it is up to the CSP to actually offer and implement these features correctly. As stated in section 4.1.2 the CSP's host machines cannot be trusted and as such the security capabilities provided by it cannot be trusted either. Hence this approach does not provide customers with the appropriate protection.

Unified threat management based approach

This approach uses a UTM in order to inspect and filter all incoming packets before allowing them to reach the host. In essence it acts as a modern firewall performing deep packet inspection (hopefully with knowledge of the packet's context) in order to detect harmful traffic before they reach the host [20]. This approach can help handle attacks which would allow an attacker to extract customer data, such as SQL injection, Cross site scripting (XSS), etc.

However according to the threat model we defined, UTM is not enough. UTM is placed between the TaaS solution and outside traffic, it works under the implicit assumption the CSP can be trusted. As stated by our threat model this is not an assumption we can make. Hence this approach does not provide customers with appropriate protection.

On-demand security approach

This approach allows customers to provision VM security capabilities according to the threat model defined and created in this approach. The security capabilities deployed on a specific VM are collected and monitoring by the solution and changed if the security requirements of the VM are changed or if the server is under attack. According to [19]:

"A malicious cloud manager can compromise the security of a customer's code and data. A cloud manager can compromise the security of a VM by allocating it to the wrong set of servers. A cloud manager may also run at a higher privilege and access a server's memory, thereby breaching a VM's confidentiality and integrity".

Hence applying the Cloud manager protection security level should answer the security concerns raised by a TaaS solution. However, according to the threat model we defined, the host cannot be trusted. Hence this approach does not provide customers with appropriate protection.

4.3.1.3 Cryptographic cloud storage approach

The cryptographic cloud storage approach defines a system which achieves customer control over the data stored, data integrity, data confidentiality and data non-repudiation via the use of encryption. The article [21] then goes on the described the way the data can be ingested and encrypted in order to allow for search over the encrypted data.

However this approach does not provide for the protection needed for a fully functional TaaS solution. As stated in section 3.3.3 the customer data is stored in encrypted form within the CSP's infrastructure. According to [17] the current encryption techniques are only functional for data at rest and data in transit. It is not possible to perform operations on encrypted data – aside from simple sum and some products. Hence, while test results can be stored in an encrypted form and then transmitted to the customer and the data would still remain confidential, performing operations on encrypted customer data is difficult.

Utilizing a fully homomorphic encryption can allow us to perform the above mentioned operations. A fully homomorphic encryption scheme is [34] [35]:

"...given ciphertexts that encrypt $\pi 1, \ldots, \pi t$, fully homomorphic encryption should allow anyone (not just the key-holder) to output a ciphertext that encrypts $f(\pi 1, \ldots, \pi t)$ for any desired function f, as long as that function can be efficiently computed. No information about $\pi 1, \ldots, \pi t$ or $f(\pi 1, \ldots, \pi t)$, or any intermediate unencrypted values, should leak; the inputs, output and intermediate values are always encrypted."

The process of constructing a fully homomorphic encryption scheme and the performance of these schemes are further discussed in [34] [35].

It is important to understand the scheme presented in [34] is not practical as the ciphertext size and computation time increase sharply as one increases the security level. For example: In order to obtain 2k security against known attacks, the computation time and ciphertext size are high-degree polynomials in k. The approach described in [35] attempts to simplify the previous approach and make it more efficient; however it only manages in simplifying to concept.

Currently there are several working implementations of fully homomorphic encryption scheme which are presented in [36] and [37]. An open source implementation of a fully homomorphic encryption scheme called "HELib" is published on GitHub [38].

We can see how fully homomorphic encryption schemes can solve the problem of running testing suites on encrypted data. However fully homomorphic encryption schemes are

computing-intensive [17], this causes issues from other vital processes such as development, debugging and validation of software working with encrypted data. As such this approach does not satisfy the functional requirements defined in section 4.2.

4.3.1.4 Trusted computing approach

The trusted computing approach seeks to resolve security issues by taking advantage of security designated hardware component and modifying software applications to take advantage of them.

Specification for Trusted Computing Module (TPM) [22] [23], a cryptoprocessor which can be used to protected applications from threats posed by external entities, have been published by the Trusted Computing Group (TCG) Module. These specifications have been implemented by leading hardware vendors such as Intel, AMD etc. As they are currently implemented there are several issues which prevent TPMs from being used by cloud applications:

Different implementations provide different functionality

TPM is an open standard, hence several implementations exist and among them are Intel TXT and AMD. According to [23] each of these implementations supports different security features, because of this issue developing operating systems which takes advantage of TPM is difficult.

Lack of parallel execution support

According to [23] the current architecture of TPM processors only supports one secure execution environment at a time, even on multi core systems. This means that if a core is not part of a secure execution environment, it will be halted while a secure execution environment is exists.

Hence, the TMP architecture does not support a use case where several tenants (in our case several test case executions), require concurrent independent secure execution environments. The TMP architecture does not support a basic cloud computing use case, performing multi-tenant concurrent execution.

The extension of the Intel TXT solution presented in [23] provides the ability to deploy multiple, parallel secure execution environments on multicore processors. It allows for one secure environment per VM rather than per system, and allows each application to dynamically switch

between normal VM execution and running a sensitive application within a secure environment concurrently. This is accomplice by utilizing a disengaged hypervisor which is responsible for VM startup and shutdown and does not interact with them at run-time. However, it requires altering the TPM processor preventing us from deploying our solution of commodity hardware. This would force the solution to be deployed on custom hardware, restricting growth.

As we can see while TPM processors can be used to support the TaaS solution we wish to implement, there exists several functional and technical issues which prevent the use of a TPM processor to satisfy the functional requirements raised in section 4.2.

4.3.1.5 Virtual TPM approach

The virtual TPM approach seeks to provide TPM levels of security by virtualizing the TPM and deploying our solution within these VMs. This would allow applications deployed within these VMs to take advantage of TPM levels of protection without actually having TPM processors deployed on the hosts; this would in turn allow our TaaS solution to be deployed on any virtualization enabled host machine.

If we look into the security analysis provided in [25], we can see that current virtual TPMs offering rely on physical TPM in order to provide protection:

***Xen* virtual TPM**

As we stated in section 3.3.5 the virtual TPM implementation provides by Xen creates domain building component which are measured by a physical TPM when it is booted. The component is responsible the creation and destruction of critical domain and pairing virtual TPM instances with virtual machines.

The Xen virtual TPM implementation is dependent on a physical TPM processor deployed on the host as it relies on measurements provided by the TPM on the domain building component at bootstrap. Furthermore, there is not authenticated link between VMs and corresponding virtual TPMs; as such any VM can be paired to any virtual TPM.

Lastly, if the physical TPM's drivers are installed on Dom0 a malicious CSP admin may have access to the key being used to encrypt the virtual TPMs which doesn't allow for extension of the Chain of Trust from the host to the virtual TPM.

***QEMU* virtual TPM**

As stated before the virtual TPM implementation provided by QEMU is either a direct mapping of a physical TPM (QEMU Passthrough) or a fully virtualized TPM which does not rely on a physical TPM processor deployed on the host machine.

In regards to QEMU Passhrough, any security capabilities provided by it are directly linked to the underlining physical TPM, hence any breach in that TPM would cause a breach in the virtual TPM. In regards to the fully virtualized TPM, this implementation depends on dedicated encrypted image file to contain its secrets. However the password protecting this image file is not strong enough and there are no strong links between the image file, the virtual TPM and the corresponding VMs.

As we can see virtual TPM implementations are not yet mature enough and such cannot be used to support the TaaS solution we wish to implement. The QEMU and XEN virtual TPM implementation provide base for future secure implementation, currently they depends on the TPM processor deployed with the physical host for security. As we have shown in section 4.3.2.4 a physical TPM processor cannot be used for protection.

4.3.1.6 Intel Software Guard Extension approach

The Intel Software Guard Extension (SGX) approach is designed to increase the security of software through an "inverse sandbox" mechanism. In this approach legitimate software can be sealed inside an enclave and protected from access by the external entities regardless of the privilege level of the latter.

While it seems that by utilizing Intel SGX we will be able to implement our TaaS solution with sufficient security there are a several issues which prevent us from utilizing SGX as the sole source of security in our implementation. According to [31] and [32]:

- SGX was designed in order to allow applications to run specific parts of their code inside an enclave. This requires an application to be written with SGX in mind and would prevent the use of 3rd party components (such as Apache Ant or SQL server) to protect sensitive areas of their application code.
- Running entire applications inside an enclave presents several problems for SGX. Applications perform several operations which SGX was not designed to support: Runtime code and data loading, dynamic memory allocation, virtual memory protection changes, usage of thread-locale storage, execution of user mode instructions, exception handling (page faults, divide-by-zero, floating point etc.)

Furthermore there are a few security issues we must take into account:

- As stated in section 3.3.6 remote attestation proves that application code is executed by a SGX processor, however it does not guarantee that the processor is located in the CSP's infrastructure. In this case an attacker could purchase a SGX processor and extract the processor's master secret. They can then impersonate a SGX processor within the CSP's infrastructure.
- According to [39] SGX does not discuss how secure user input and output is handled from inside an enclave. Furthermore it makes claims that such technology does not yet exist. As such we must ensure that any user I/O (test case creation, test data upload) is not done from inside an enclave. Other ways to protect this information should be used.
- According to [31] applications executed inside an isolated environment, in our case an enclave, are vulnerable to attacks by a malicious OS. The OS can perform "Iago attacks" which return semantically-incorrect results from system calls and may seek to exploit latent bugs within the application. According to the threat model defined in section 4.1.2 we cannot trust the OS.

Furthermore there are several technical issues which prevent the current SGX implementation from being used for the deployment of whole applications.

4.4 Solution design

This section focuses of the design phase, where the solution is designed to satisfy the requirements identified in the previous phases. The requirements identified in the requirements analysis phase are transformed into a system design document that accurately describes the design of the system and that can be used as an input to system development in the next phase.

For our design we will be using UML diagrams which are standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system using graphic notation.

4.4.1 Functional design

In this section we will perform our first design iteration, focusing on the functional requirements of this solution.

The functional design (behavior and structural) of TaaS solution is detailed in section 2.2 and [12]. In short the TaaS solution is deployed as public cloud using Xen VMs on physical machines (with no hardware specifications) with customer facing interfaces. The solution is composed of the following layers:

- ***Test service tenant layer***: This layer allows QE personnel to design and implement testing suites. It also allows service contributors to ingest testing services and VM images.
- ***Test case management layer***: This layers processes test suites (test cases and test data), acts as a Testing service bus (TSB) which clusters and dispatches scheduled testing tasks to corresponding VMs, monitors test case execution on VMs and provides tenants with test results.
- ***Test resources management layer***: This layer acts as an infrastructure to the TaaS application. It monitors and allocated physical machines and VMs according to testing needs. It also provides various cloud solution infrastructure services such as billing, provisioning, user management, etc.

- ***Test case layer***: This layer represents a test case instance. Each instance is responsible for creating a workflow for testing operations performed during execution, providing the various testing services needed for execution, executing the test case workflow and test result aggregation.
- ***Testing storage layer***: This layer stores test cases, test data, VM images and testing results.

A diagram detailing the solution's composition and behavior has been presented in section 2.2 by figure 2.

4.4.2 Security design

In this section we will perform our second design iteration, focusing on the security requirements of this solution.

We will alter the design presented in section 4.4.1 to fulfil the security requirements raised in section 4.2 which focus on data protection and integrity. We will discuss what technologies will be used and why, present additional components created in order to provide adequate security and refactor the existing solution.

4.4.2.1 Technologies used in solution

It this section we will detail the various technologies (algorithms, protocols, methodologies and components) employed in our solution.

Intel SGX

Under the threat model we defined in section 4.1.2 the host's processor is implemented correctly and is not compromised. We can use the Intel SGX processor as a root of trust and dynamically extend trust from it to our entire solution. This solution will be deployed in Xen VMs using hosts which contain a SGX processor.

Encryption algorithm

In order to perform data encryption and decryption we must choose an encryption algorithm. Encryption algorithms are divided into 2 categories [30]:

- ***Symmetric-key encryption*:** Uses the same key for both encryption and decryption. In order to maintain confidentiality the symmetric key should only be known by the entity which encrypts the data and the entity which decrypts it. This can cause a scalability issue: if there are n users, each user would need to n-1 symmetric keys in order to communicate securely with the rest of the network.
- ***Asymmetric-key encryption*:** Uses different keys for encryption and decryption.
  Bob has one private key used for the decryption and one public key used for encryption, messages encrypted using the public key can only be decrypted using the private key and an attacker cannot ascertain the private key from the private one. If Alice wants to communicate with Bob, Alice will use the public key to encrypt the message and then Bob will use his private key to decrypt the message. There is no scalability issue as each user only needs to publish one public key in order for others to communicate with him. However asymmetric-key encryption is usually slower and weaker.

According to [30] symmetric key encryption has the advantage in our scenario. It is faster and more efficient overall and as long as proper key management is maintained it is secure. In order to handle the key distribution issue we raised, we will use a secure communication protocol which will be detailed later.

According to [40] the Blowfish encryption algorithm is the fastest and most secure block cypher symmetric key encryption available today. Hence we will be using Blowfish in order to encrypt and decrypt all customer data.

Data verification

We will use a proof of storage protocol in order to verify that the data has not been modified while it was stored. A proof of storage protocol is [21]:

"A protocol executed between a client and a server with which the server can prove to the client that it did not tamper with its data."

Proof of storage protocol can be privately or publicly verifiable, we will use a publically verifiable protocol as it allows the both the customer and the solution to verify that the data has not been tampered with. Proof of storage protocols can be executed several times and the amount of data exchanged between the challenger and the verifier is extremely small and independent of the size of the data.

Different proof of storage protocols exist [41] depending on whether the data is static or dynamic. In our use case changing the test cases and test data would constitute as new testing scenarios and not as an update of the existing one; test results are static by nature as they represent the outcome of specific test case execution. Hence our will we use a static data proof of storage protocol.

Data verification will be done using the Shacham and Waters proof of storage algorithm which is detailed in [41]. The algorithm inserts random value blocks into at random positions within the encrypted data, an error correction code is applied to the data and an authenticator is applies to each file block. The data and authenticators are then stored.
In order to verify that data correctness, the challenger sends a random list of indexes (corresponding to the authenticators) to the verifier who sends the corresponding response. The challenger then analyses the response to verify data correctness.

Secure communication protocol

In order to facilitate secure communication between components, secure communication channels need to be established. We will use the Transport Layer Secure (TLS) protocol to establish these channels.

According to the specification published by OWASP [42], TLS provides application communication with: Protection from unauthorized discloser, unauthorized modification, integrity guarantees and replay prevention of communication between clients and servers. TLS should to be used when transmitting sensitive data. We use TLS 1.2 as it is currently the most secure protocol, as security breaches have been found in SSLv3.

TLS uses private/public key pairs [43] and a handshake protocol in order to establish a secure communication channel between server and clients. As states in section 3.3.6 a SGX processor provides a strong private/public key pair which can be used for this purpose.

Cloud quoting enclave

In section 4.3.2.6 we have shown that remote attestation does not guarantee that the attesting SGX processor is part of the CSP's infrastructure. An attacker could purchase a SGX processor and extract the processor's master secret after a long term attack. They can then impersonate a SGX processor who is part of CSP's infrastructure.

We will use the principles presented in [32] to handle this issue. An additional quoting enclave is created by the CSP when a SGX machine is added to the datacenter. Once it is created the enclave's public key is published and private key is sealed inside the enclave. The cloud quoting enclave is used to generate quotes in conjunction is to the SGX processor's quoting enclave. This would allow us to verify that code is executed on hardware owned and certified by the CSP and is part of a certain datacenter.

Haven execution environment

In section 4.3.2.6 we have shown that executing 3rd party binaries, which were not created with SGX in mind, inside an enclave causes a several problems. Applications perform several operations which SGX was not designed to support: Runtime code and data loading, dynamic memory allocation, virtual memory protection changes, usage of thread-locale storage, execution of user mode instructions, exception handling (page faults, divide-by-zero, floating point etc.). Furthermore Running an entire applications inside an enclave exposes them to attacks by a malicious OS.

According to section 2.2 executing test cases is equal to running an entire application as application functionality, application scalability and end to end scenarios are tested.

We will use the principles presented in [31] to handle this issue. Microsoft Haven is a prototype which utilizes Microsoft's Drawbridge, LibOS and Intel's SGX to create a shielded execution

environment for unmodified applications. It's is implemented on Windows 8.1 using a custom built driver.

Drawbridge is a virtualization container which provides secure isolation, persistent compatibility and execution continuity, with drastically lower resource overheads. Drawbridge utilizes two core technologies:

- ***Picoprocess*:** A process-based isolation container with a minimal kernel API surface.
- ***Library OS*:** A version of Windows which is modified to run efficiently inside a picoprocess.

The Haven execution environment composition is displayed in figure 3 which is taken from [31]. Haven adds two layers to Drawbridge:

- ***Shield module:*** Located inside the enclave and positioned below LibOS, it provides a LibOS abstraction. It is responsible for protection of the LibOS and the application from attacks by a malicious host OS (as described earlier). This is done by validating all communication between the host and LibOS.
- ***Untrusted runtime*** Located outside the enclave. Responsible for creating the enclave, loading the shield module and forwarding calls between the enclave and host OS.

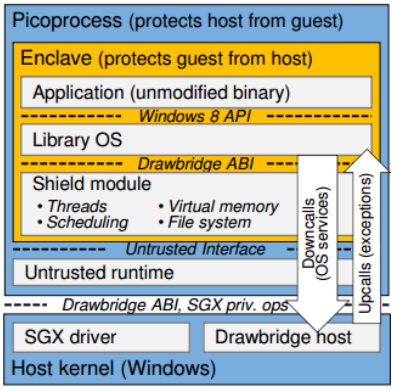

Figure 3Haven Execution Environment

By using the Haven execution environment we can execute an entire test case instance and provide the protection and functionality needed for test case execution.

The Haven component does not support VM migration. In order to facilitate VM migration the Drawbridge ABI will need to be augmented using the checkpoint functionality which is proposed in [44]. When a VM is suspended for migration the entire environment running inside the

enclave will be sealed according to the sealing identity and stored in the unprotected memory. Once the VM is migrated and restarted an enclave will be started, the sealed environment will be moved to the new enclave, unsealed and used in order to resume execution.

4.4.2.2 Secure solution components

In this section we will detail what are the components which we be added to our solution. These components will utilize the technologies detailed in section 4.4.2.1.

Enclave verification component

EnclaveVerifier is responsible for enclave attestation verification. It will verify if reports or quotes originate from an enclave executing on a SGX processor who is part of the CSP's infrastructure or the same host. The EnclaveVerifier contains the following functions:

- ***verifyQuote(quote, origin):*** This function verifies that the quote originated from a host containing a SGX processor who is part of the CSP's infrastructure, using the public keys published by the cloud quoting enclave and the host's quoting enclave. The function returns true if and only if the attestation succeeds.
- ***verifyReport(report, origin):*** This function verifies that the report originated from an enclave which is executed on the local machine. The function returns true if and only if the attestation succeeds.

Figure 4EnclaveVerifier class

Enclave executing component interface

EnclaveExecuting is an interface which must be implemented by every component which is executed inside an enclave. It is responsible for returning the enclave's measurements, creating a report using another enclave's measurements for local attestation and creating a quote for remote attestation. EnclaveExecuting contains the following functions:

- ***getMeasurements()*:** Returns the current enclave's measurements.

- ***createReport(targetEnclave)*:** Receives a target which implements EnclaveExecuting and creates a report using its measurements.
- ***createQuote()*:** Create a quote for the current enclave using the quoting enclave and the cloud quoting enclave.

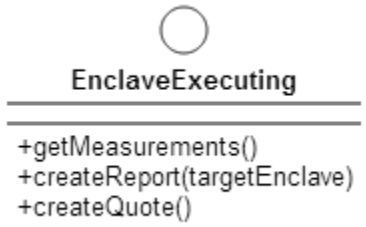

Figure 5EnclaveExecuting interface

Bootstrapble component binary

BootstrapbleBinary represent a component which will be executed inside an enclave. It contained encrypted compile binary of the component; it implements the EnclaveExecuting interface and contains the following functions:

- ***execute()***: Verifies the encrypted binary correctness, retrieves the encryption key, decrypts the encrypted binary and launches the component binary.

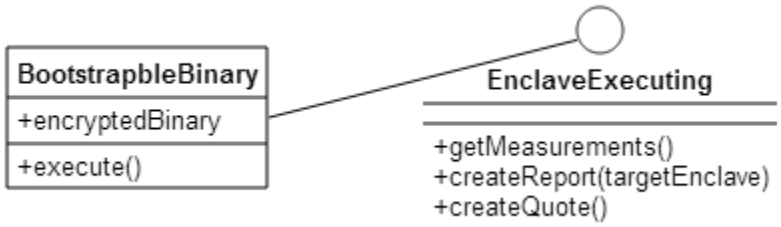

Figure 6BootstrapbleBinary

Enclave bootstrap loader

EnclaveBootstrap is responsible for retrieving BootstrapbleBinary from the testing storage layer into an enclave. EnclaveBootstrap contains the following functions:

- ***bootstrapComponent(componentID)*:** This function receives a component ID, retrieves the corresponding BootstrapbleBinary, creates an enclave and loads the binary into it.

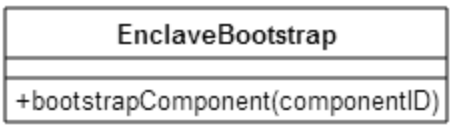

Figure 7EnclaveBootstrap

Secure enclave communication controller

SecureEnclaveCommunicationController is responsible for secure communication between enclaves and external entities. It executes inside an enclave and acts as a quoting enclave for the purpose of remote attestation. It provides a quote to entities outside of the host, verifies enclave reports, opens secure communication channels to outside entities using TLS, opens secure communication channel to other enclaves using TLS and forwards request from and to the other enclaves. SecureEnclaveCommunicationController contains an EnclaveVerifier instance and the following functions:

- ***challenge()*:** The function provides a quote for the enclave in whom the controller is executed in.
- ***attestQuote(quote, origin)*:** The function receives a quote from an enclave. It then attempts to verify the quote using the EnclaveVerifier. If attestation is successful a secure connection is opened between the enclave and the controller. If the attestation fails an exception is raised.
- ***attestReport(report, origin)*:** The function receives a report from an enclave. It then attempts to verify the report using the EnclaveVerifier. If attestation is successful, a secure connection is opened between the controller and the enclave. If the attestation fails an exception is raised.
- ***handleRequest(request)*:** The function receives a request through a secure channel. The request contains the request final target, request operation and request data. If a secure connection between the controller and target exists the request will be forwarded and it the response will be forwarded back. If a secure channel does not exist an exception will be raised.

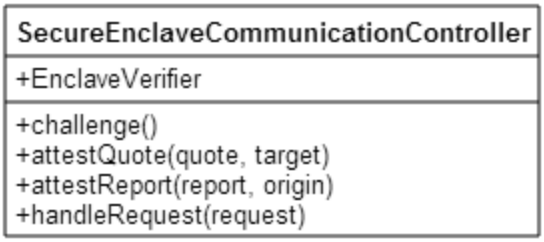

Figure 8SecureEnclaveCommunicationController

Data encryption component

DataEncryptionModule is responsible for data encryption and decryption using the Blowfish symmetric-key encryption algorithm. DataEncryptionModule contains the following functions:

- ***encrypt(data, key):*** Receives a stream (representing the unencrypted data) and a char array (representing the encryption key) and outputs the encrypted data.
- ***decrypt(data, key):*** Receives a stream (representing the encrypted data) and a char array (representing the decryption key) and outputs unencrypted data.

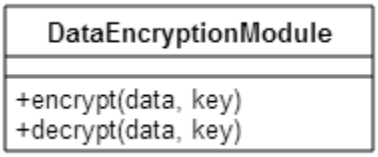

Figure 9DataEncryptionModule

Data correctness component

DataCorrectnessModule is responsible for processing encrypted data using Shacham and Waters proof of storage algorithm and verifying data correctness using the same algorithm. DataCorrectnessModule contains the following functions:

- ***process(data):*** Receives a stream (representing the unencrypted data) and outputs the data encoded for verification.
- ***verify(data, challenge):*** Receives a stream (representing the encoded data) and an index array (representing the challenge) and outputs a verification response.

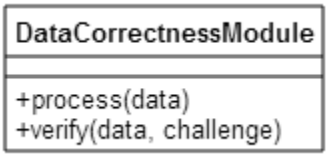

Figure 10DataCorrectnessModule

Key management component

The DataEncryptionModule requires symmetric encryption keys in order to protect customer data. As stated in section 4.1.2 the CSP infrastructure cannot be trusted and unencrypted keys should not be managed there. Instead keys will be managed by a KeyManager which is executed inside an enclave. This component will be responsible for: generating symmetric encryption keys for new tenants, storing them and supplying them to the appropriate entities.

In order to protect the keys from being lost if the host crashes, keys will be sealed using the enclave's sealing key who value is set according the sealing identity policy, encrypted key will then then be stored in an unprotected persistent database.

If customers wish to maintain control over the encryption key, they can implement their own key management server on premise and provide an access URL and authentication credentials upon registration to the TaaS solution. KeyManager contains the following functions:

- ***getEncryptionKey(tenantID, type)*:** Receives a tenant ID and data type, if the key datastore already contain an associate encryption key it is returned. Otherwise it generates a symmetric encryption key, encrypts it, stores it within the datastore and returns the generated key.

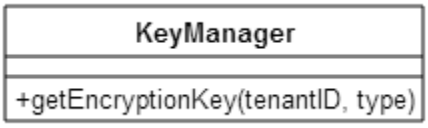

Figure 11KeyManager

Enclave executables processor

ExecutableProcessor is responsible for processing components and VM images for executing within our solution. It is executed inside an enclave; however unlike the other components which are executed inside an enclave it is stored within the testing storage layer after it is was already processed for bootstrapping. ExecutableProcessor contains a DataEncryptionModule and the following functions:

- ***processComponent(componentID, componentBinary)*:** This method receives a component binary, encrypts it using the DataEncryptionModule, creates a BootstrapbleBinary and stores it in the testing storage layer using the componentID.

- ***processVMImage(imageDetails, vmImage)*:** This method receives a VM image, bundles it with the LibOS binary and stores it in the testing storage layer using the image details.

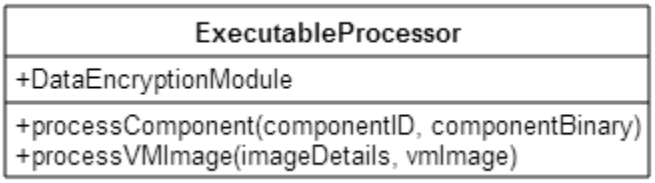

Figure 12ExecutableProcessor

Haven Controller

HavenController is responsible for creating a Haven execution environment and a VM within it. HavenController contains the following functions:

- ***launchVM(imageID)*:** Receives a VM image ID and launches it within a Haven execution environment.

Figure 13HavenController

4.4.2.3 Solution refactoring

In order to protect customer data we must understand what layers use it and how. By analyzing the solution behavior specified in sections 4.4.1 and 2.3 we can see that:

- ***Test data*:** Test data is used in the following cases
  - ***Test case management layer:*** Test data is ingested in the test case management layer where the data is uploaded by the test tenant, processed and stored within the testing storage layer.
  - ***Test case layer:*** Test data is used by the test case which is executed on a VM within the test case layer.
- ***Test cases*:** Test cases are used in the following cases
  - ***Test case management layer:*** Test cases are ingested in test case management layer where the solution's capability to run them is evaluated, tests are clustered and stored within the testing storage layer.
  - ***Test case layer:*** Test cases are executed on VMs within the test case layer.

- ***Test results*:** Test results are used in the following cases
  - ***Test case layer:*** Test results are collected within the test case layer by the result reduction sublayer and stored within the testing storage layer.
  - ***Test case management layer:*** Test results are supplied to test tenants through the task monitor component within the test case management layer.

We will use the components defined in section 4.4.2.2 when modifying the solution.

Test case management layer

The test management layer receives customer data from the test service tenant layer and provides test result to the same layer. The test case management layer also stores and retrieves data from the testing storage layer. We will refactor the test case management layer into two separate sublayers:

- ***Isolated management sublayer:*** This layer will be deployed on a host which contains a SGX processor. It is responsible for test case clustering, capability checking and test result providing.
- ***Management sublayer:*** This layer can be deployed on any host. It is responsible for test case scheduling, dispatching and monitoring.

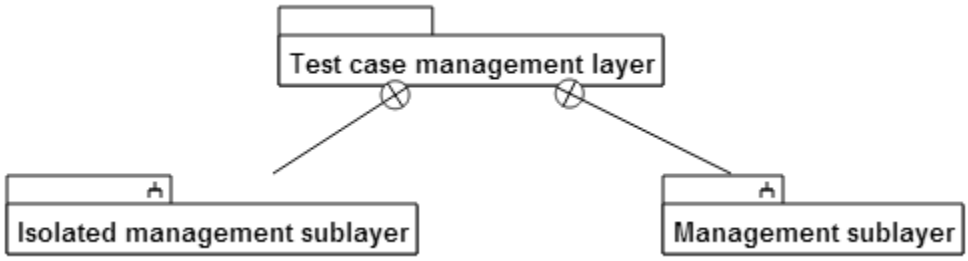

Figure 14Refactored test case management layer

The isolated management sublayer contains components which are responsible for the following functionality: testing capabilities checking, test results providing, test case clustering. Each of the components responsible for the operations stated above as well as A SecureEnclaveCommunicationController, a KeyManager and an ExecutableProcessor (which replaces the service publishing component & service repository) are stored as BootstrapbleBinary instances. An EnclaveBootstrap is also added to the layer.

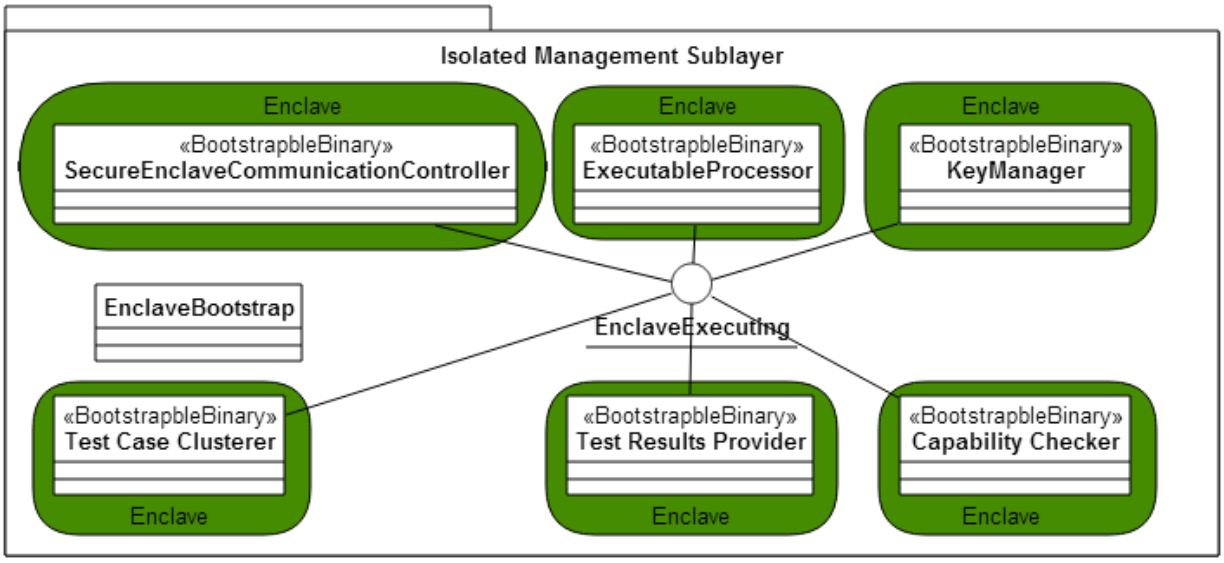

Figure 15 Isolated management sublayer

The management sublayer contains components which are responsible for the following functionality: Test case scheduling, dispatching and task monitoring. It also contains an EnclaveVerifier instance.

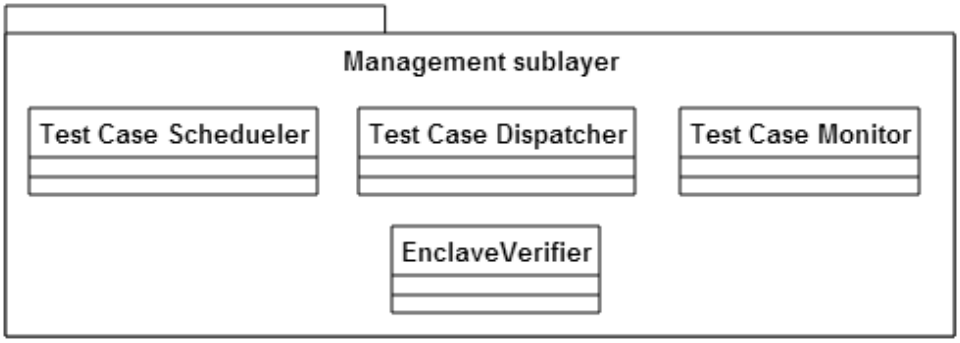

Figure 16 Management sublayer

Test resources management layer

In order to facilitate execution of the test case layer within a Haven execution environment the VM controller will need to be refactored. The VM controller is replaced by HavenController. We will also add an EnclaveVerifier to this layer.

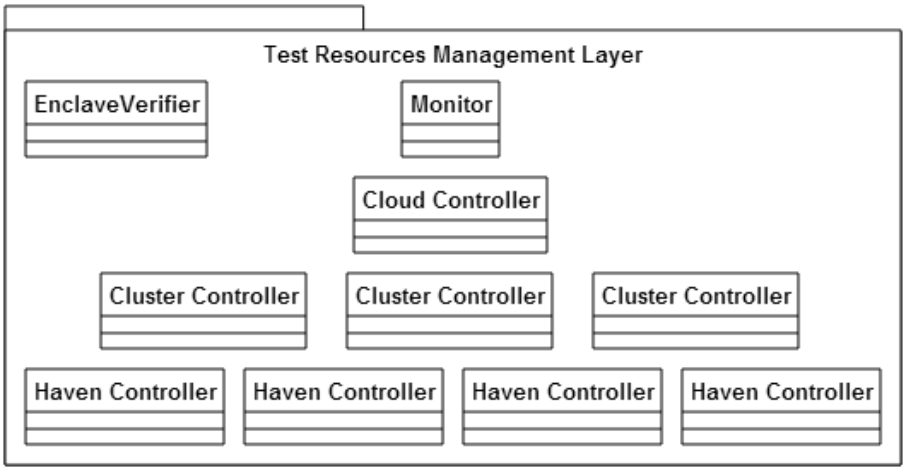

Figure 17 Refactored test resources management layer

We will need to ensure that VM images are bundled with LibOS binaries. This processed is explained later in section 4.4.2.4.

Test case layer

The test case layer performs operations within a launched VM. These include: retrieving customer data from the testing storage layer, using the data to compose an execution flow and execute it, collecting the generated test results and storing them.

As executing test cases is similar to executing an entire application and as we have shown in section 4.3.2.6 this causes additional technical and security issues. We can use the Haven execution environment discussed in section 4.4.2.1 in order to handle these issues.

The test case layer will need to be refactored for execution from inside an enclave. The Shield Module and A SecureEnclaveCommunicationController are stored as BootstrapbleBinary instances and an EnclaveBootstrap is added.

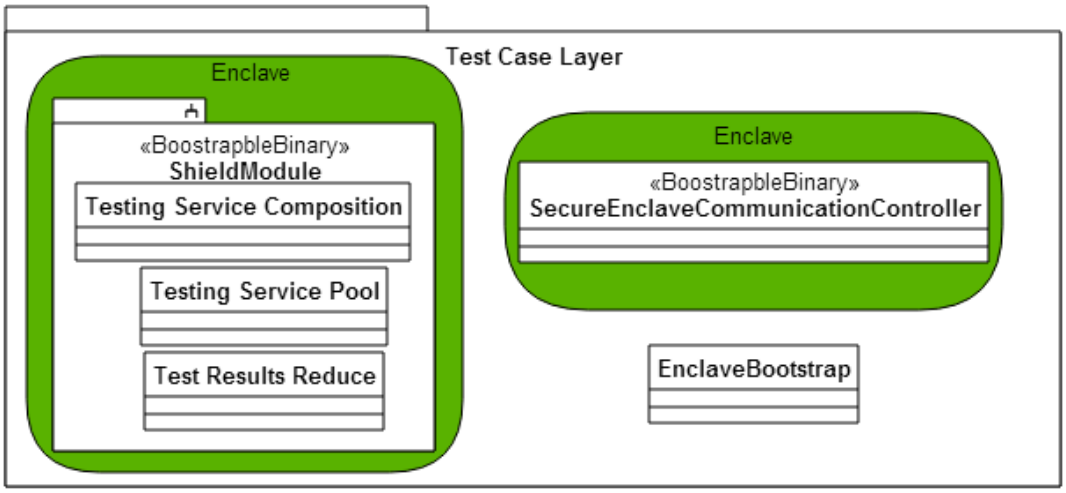

Figure 18 Refactored test case layer

Testing storage layer

The test case management and test case layers receive data necessary for their operation from the testing storage layer and store the product of the execution within the same layer. As stated in 3.3.3, in order to protect customer data within the testing storage layer it must be encrypted. We will refactor the testing storage layer into two separate sublayers:

- ***Database sublayer:*** Responsible for data storage and retrieval.
- ***Database encryption sublayer:*** Acts an abstraction for the testing storage layer. Responsible for data encryption before storage and decryption before retrieval.

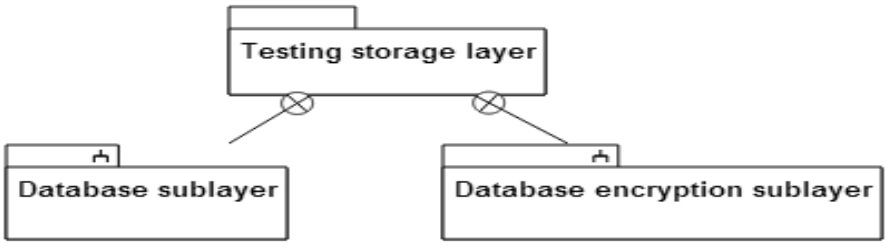

Figure 19 Refactored testing storage layer

The database sublayer contains classic database functionality: data storage and retrieval and request owner authentication. It also includes EnclaveVerifier instance.

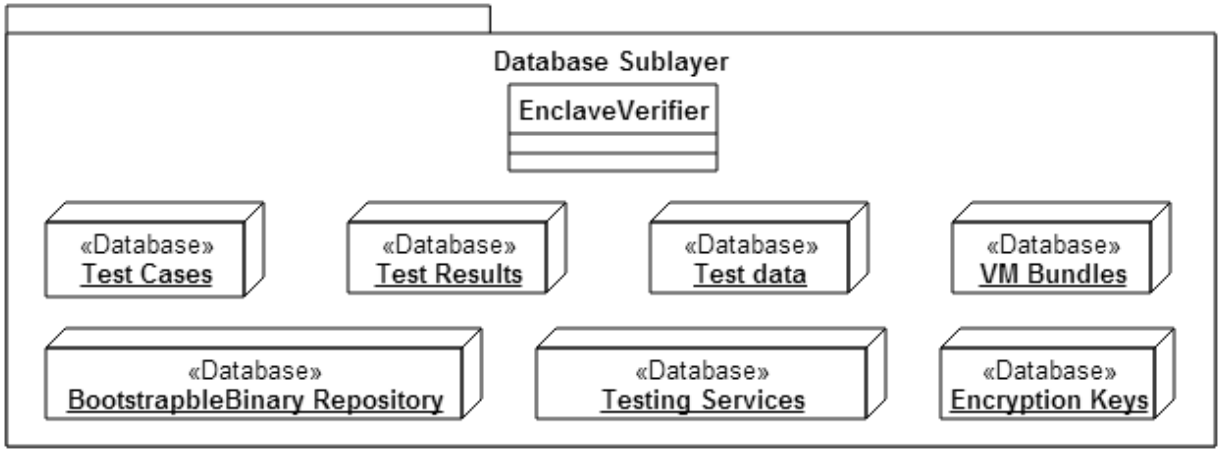

Figure 20 Database sublayer

The database encryption sublayer handles all requests directed at the testing storage layer and is responsible for data encryption, decryption and verification. A DataCorrectnessModule, a SecureEnclaveCommunicationController, and a DataEncryptionModule are stored as BootstrapbleBinary instances and an EnclaveBootstrap is added.

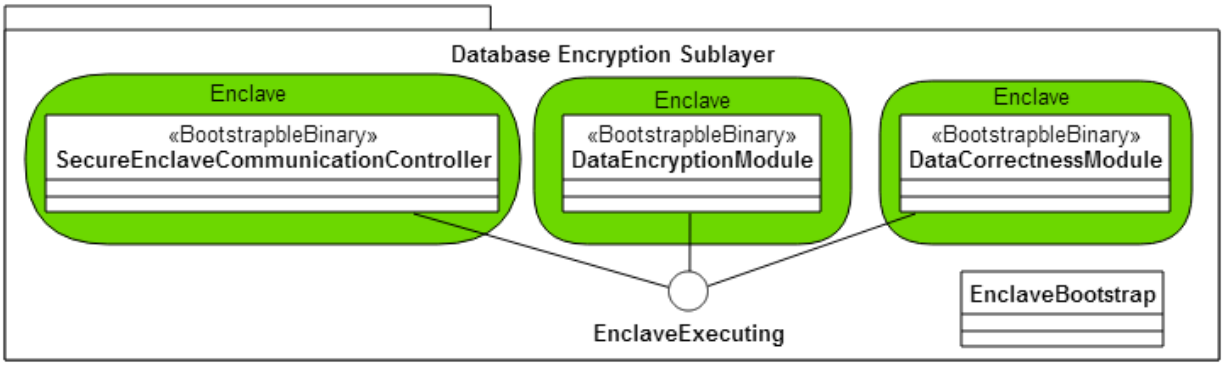

Figure 21 Database encryption sublayer

4.4.2.4 Solution behavior

In this section we will show how operations are performed in the refactored solution.

Operation from an enclave to an outside entity

We explain how operations are performed between components executing inside an enclave and a target residing outside of the enclave. The operation sequence is displayed in figure 22.

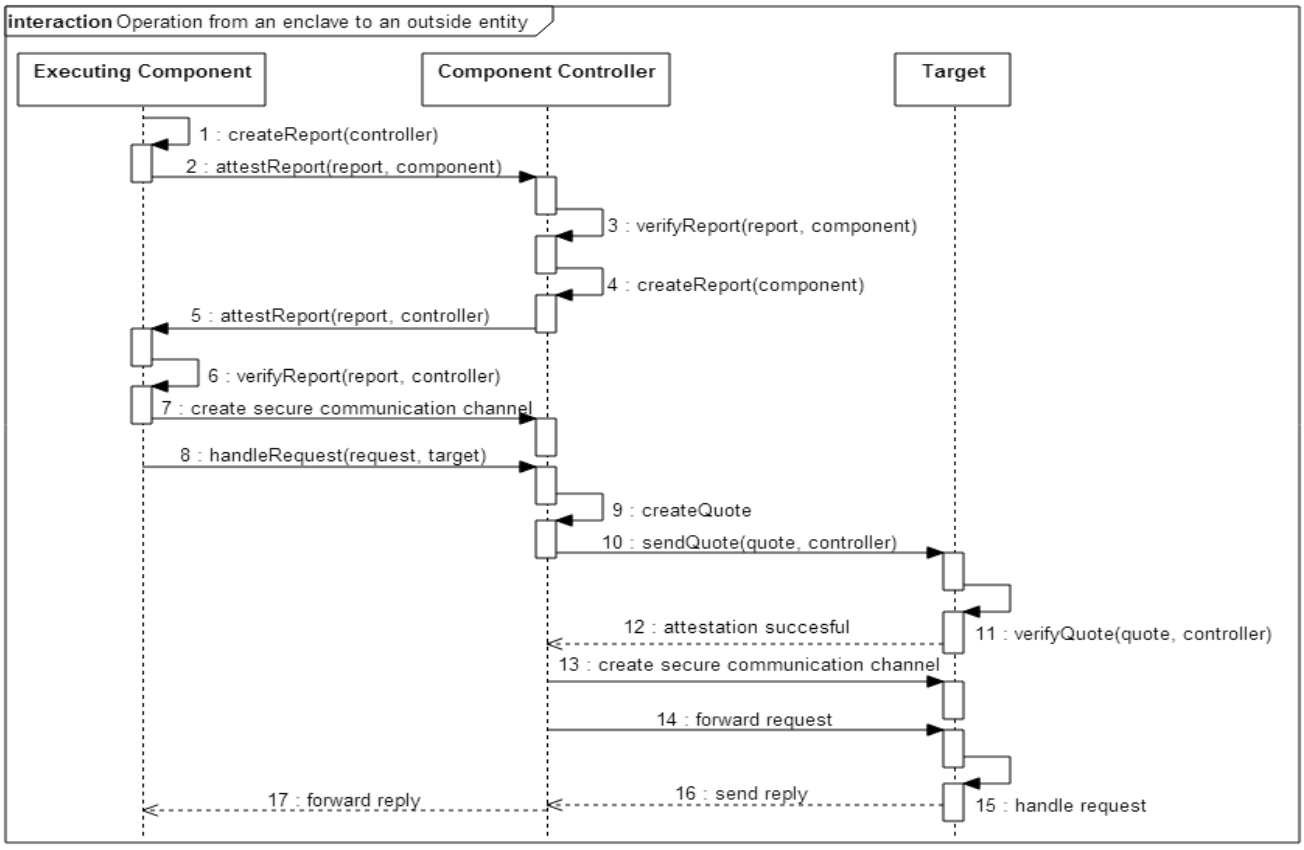

Figure 22 Operation from an enclave to an outside entity

1) ***Attestation:*** A component executing inside an enclave (the component) creates a report with the SecureEnclaveCommunicationController (the controller) measurements and sends it to the controller. The controller verifies the report using local attestation and performs the same process for the component. The component uses the enclave's public/private key pair to establish a secure connection to the controller.
2) ***Operation request:*** The component sends a request containing the request target (target), request operation and request data to the controller. The controller creates a quote and sends the quote over an insecure channel to the target. The target receives the quote, verifies it and replies to the controller.
3) ***Request handling:*** The controller uses the enclave's public/private key pair to open a secure communication channel to the target and forwards the request to the target.
   The target handles the request and replies to the controller. The controller forwards to the reply to the component.

Operation from an outside entity to an enclave

We explain how operations are performed between an entity residing outside of the enclave and an entity which executes inside of an enclave. The operation sequence is displayed in figure 23.

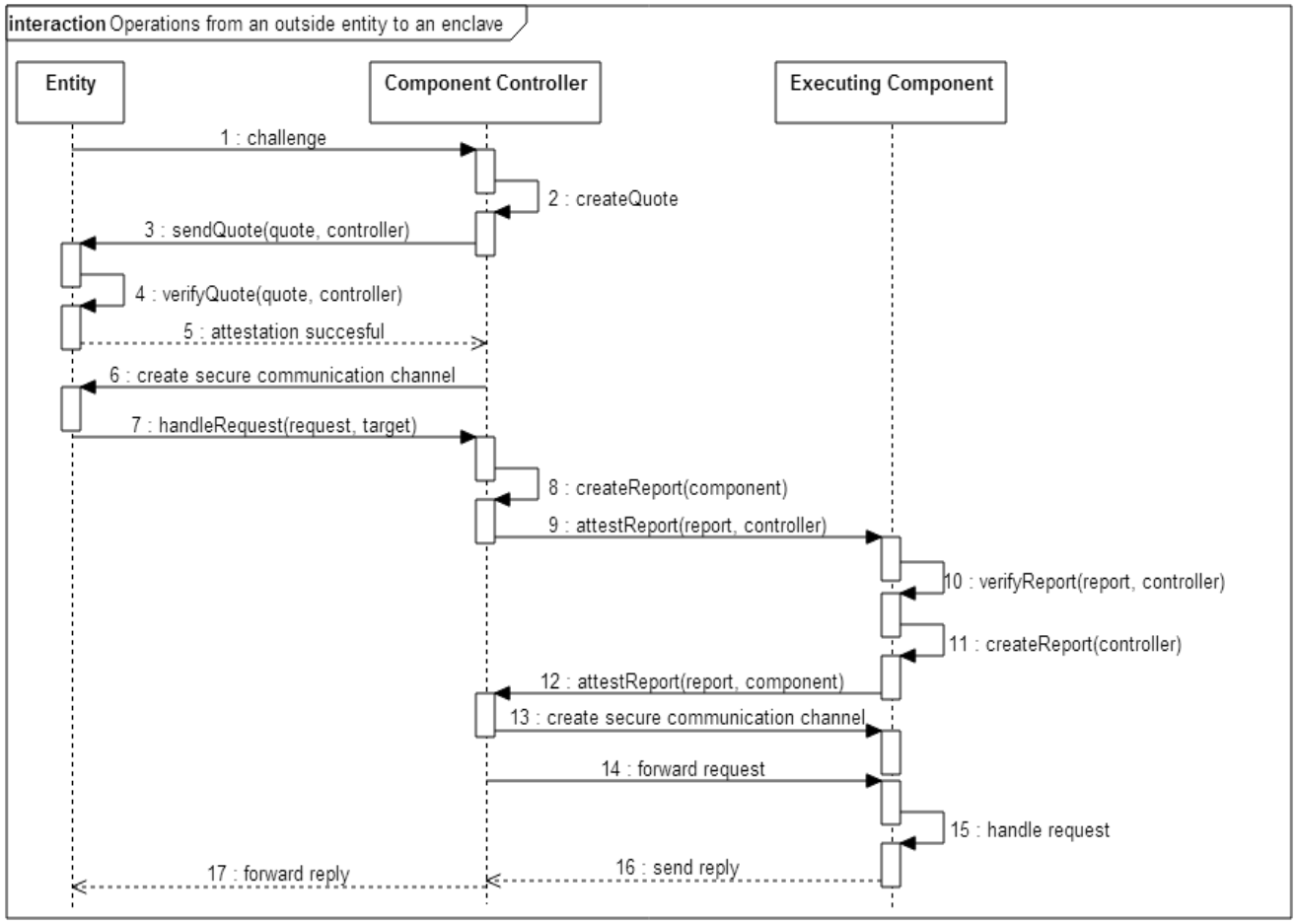

Figure 23 Operation from an outside entity to an enclave

1) ***Attestation:*** An entity executing outside of an enclave (the entity) challenges the relevant SecureEnclaveCommunicationController (the controller) to attest its identity.
   The controller creates a quote and sends it to the entity which verifies it. The controller uses the enclave's public/private key pair to establish a secure connection to the entity.
2) ***Operation request:*** The entity sends a request containing the request target, request operation, request data to the controller. The controller creates a report using the component executing inside an enclave (the component) measurements and sends it to the component. The component verifies the report using local attestation and performs the same process for the controller.

3) ***Request handling:*** The controller uses the enclave's public/private key pair to open a secure communication channel to the component and forwards the request to the component. The component handles the request returns a reply to the controller. The controller forwards to the reply to the entity.

Data storage

This section will explain how data is stored inside our solution using encryption. We have shown how operations are conducted between components inside an enclave and component outside of it, so there is no need to repeat how attestation, channel creation and request handling are performed. The operation sequence is displayed in figure 24.

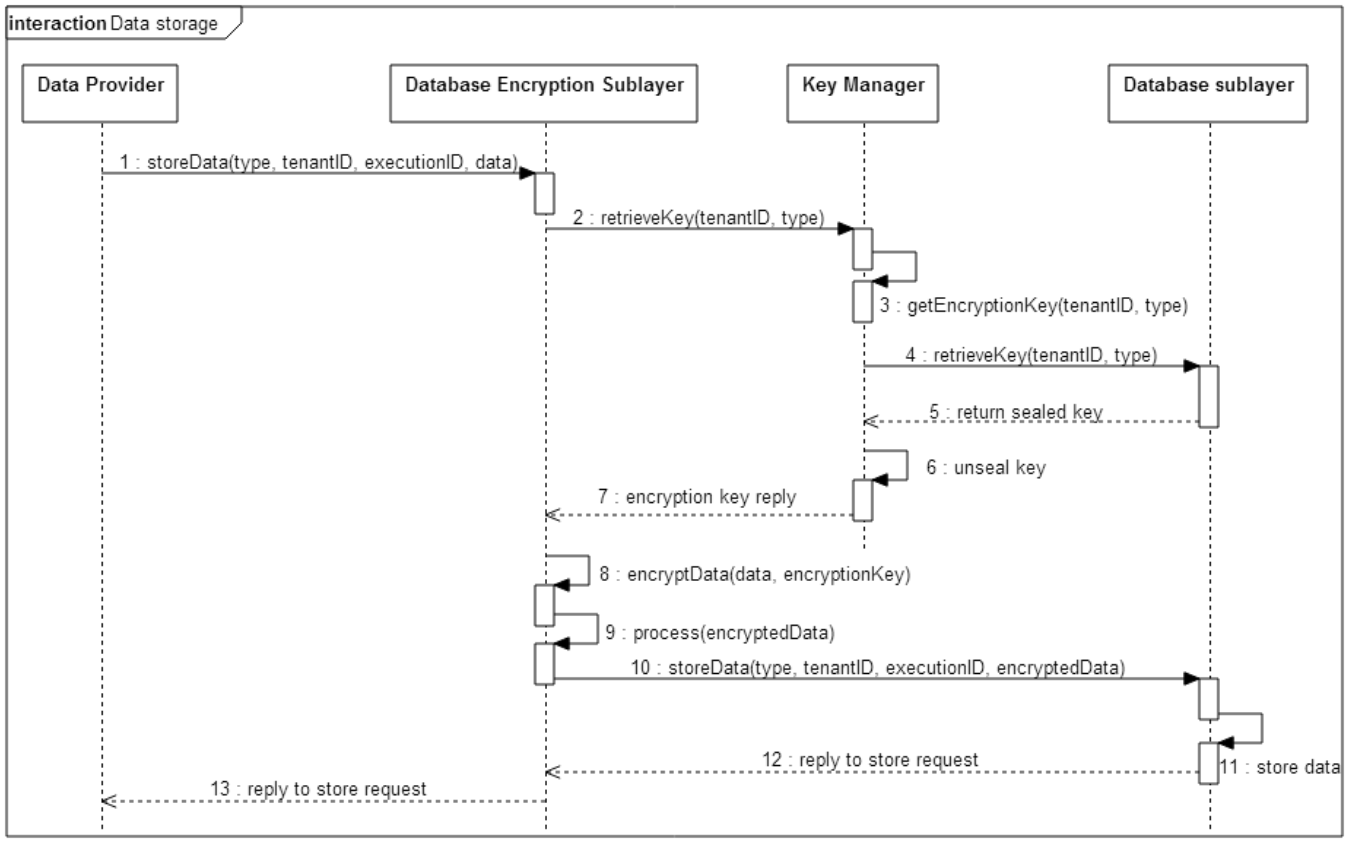

Figure 24 Data storage

1) ***Storage request:*** The component which acts as data provider (the provider) and is executed inside an enclave, sends a data store request containing the data type, tenant ID, execution ID and data to the database encryption sublayer (the encrypter).

2) ***Encryption key retrieval request:*** The encrypter creates an encryption key retrieval request contain the tenant ID, authentication data and the type of data and sends it to the key manager (the manager).
3) ***Encryption key retrieval:*** The manager checks if an encryption key for this tenant and type exists within the datastore. If it doesn't it is generated, encrypted, stored and returned. If it does exist it is retrieved from the database sublayer (the database), decrypted using the enclave's sealing key and returned.
4) ***Data processing:*** The encrypter encrypts the data using the encryption key and then encodes the data for verification purposes.
5) ***Data storage:*** The encrypter creates a data store request containing the data type, tenant ID, execution ID and the encrypted data and sends it to the database. The database stores the data in accordance to the identification data and replies.

Data retrieval

This section will explain how data is retrieved in our solution. We have shown how operations are conducted between components inside an enclave and a non-enclave component, so there is no need to repeat how attestation, channel creation and request handling are performed. The operation sequence is displayed in figure 25.

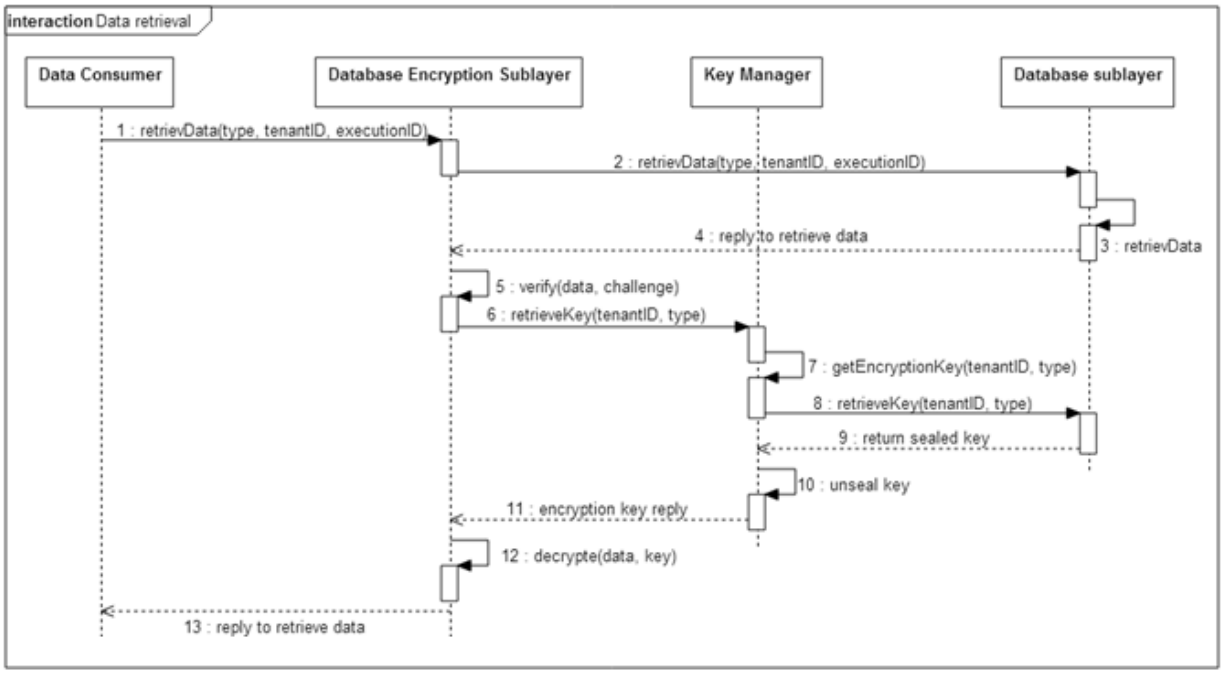

Figure 25 Data retrieval

1) ***Storage request*:** The component which acts as data consumer (the consumer) and is executed inside an enclave, creates a data retrieval request containing the data type, tenant ID and execution ID and send it to the database encryption sublayer (the decrypter).
2) ***Data retrieval*:** The decrypter creates a data retrieval request containing the data type, tenant ID and execution ID for identification and sends it to the database sublayer (the database). The database returns the data in accordance to the identification data and replies.
3) ***Data verification*:** The decrypter verifies the data correctness.
4) ***Encryption key retrieval request***: The decrypter creates an encryption key retrieval request contain the tenant ID, authentication data and the type of data and sends it to the key manager (the manager).
5) ***Encryption key retrieval*:** The manager checks if an encryption key for this tenant and type exists within the datastore. If it doesn't an exception is thrown, if a key exists it is retrieved from the database, decrypted using the enclave's sealing key and returned.
6) ***Data processing*:** The decrypter decrypts the data using the encryption key and returns in it to the consumer.

Processing entities for enclave execution

This section will explain how executables (components or VM images) are processed for execution inside an enclave in our solution. We have shown how data is stored so there is no need to repeat how data encryption, encoding and storage work. The operation sequence is displayed in figure 26.

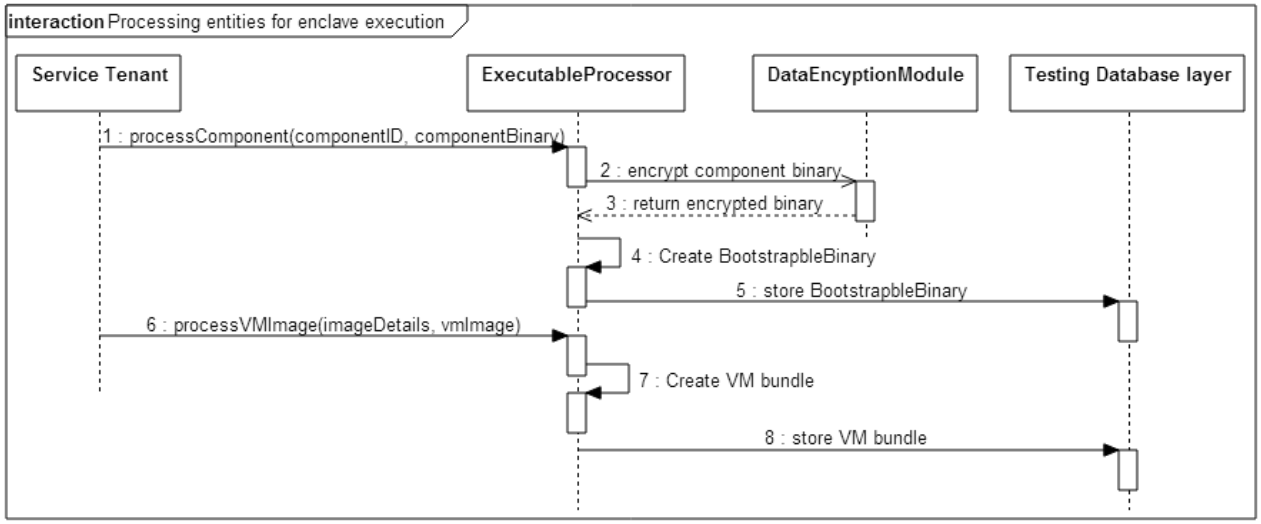

Figure 26 Processing entities for enclave execution

1) ***Executable ingestion:*** A solution developer or QE system administrator (the service tenant) sends VM image or component binary (the executable) ingestion request to the ExecutableProcessor (the processor). The processor receives the request and ingests the executable.
2) ***Executable processing:*** If the executable is a VM image, the processor bundles it with a LibOS binary and stores the bundle in the testing storage database. If the executable is a component binary, the processor retrieves the encryption key from the KeyManager, encrypts the executable, creates a BootstrappleBinary instance and stores it in the testing storage database.

Enclave component execution

This section will explain how components are executed inside an enclave in our solution. We have shown how data is retrieved, so there is no need to repeat how data retrieval, verification and decryption work. The operation sequence is displayed in figure 27.

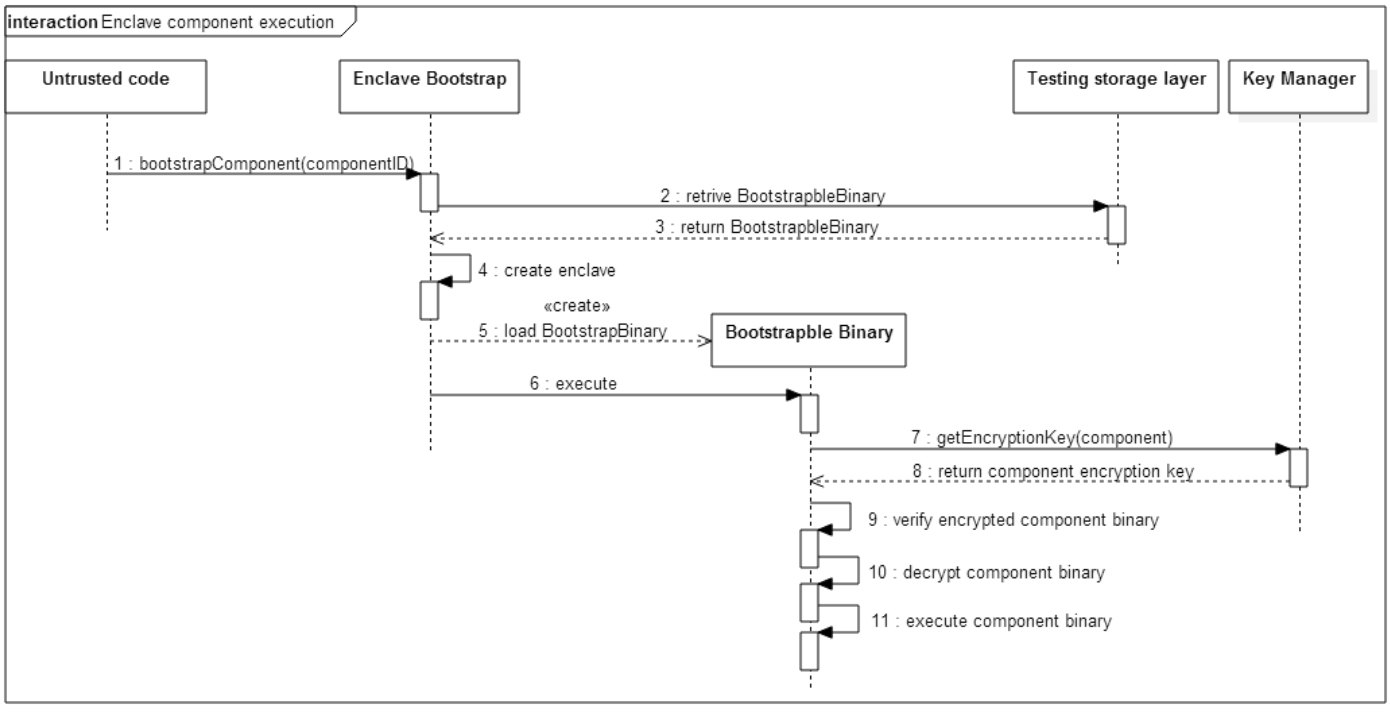

Figure 27 Enclave component execution

1) ***Component retrieval:*** The EnclaveBootstrap (the bootstrap) receives a request to load an enclave executing component (the component). The bootstrap retrieves the bootstrapble component binary (the binary) from within the testing storage layer. The bootstrap creates an enclave and loads the binary into it.
2) ***Component bootstrap:*** The binary verifies the encrypted component binary for data correctness, it retrieves the encryption key from the KeyManager and decrypts encrypted component binary. The encrypted component binary is then executed.

Test case execution

This section will explain how test cases are executed in our solution. Previously, we have shown the process of loading a component into an enclave for execution so there is no need to repeat the bootstrapping process. The operation sequence is displayed in figure 28.

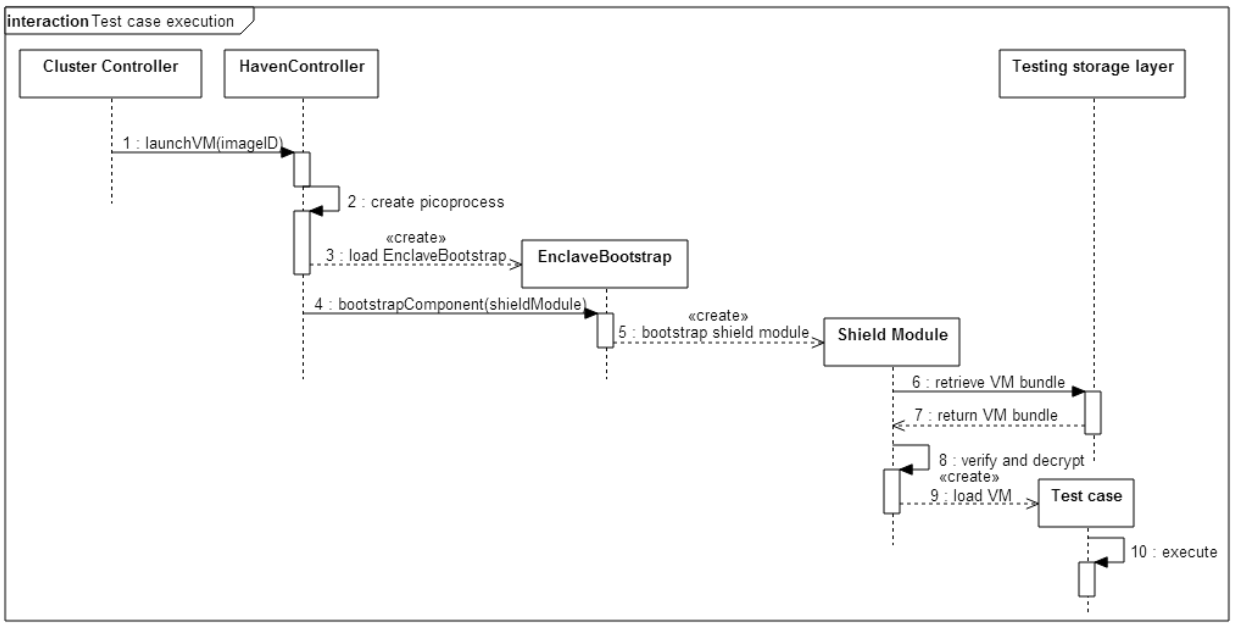

Figure 28 Test case execution

1) ***Environment creation:*** The Haven controller (the controller) receives a request to create a VM from the cluster controller. The controller uses Drawbridge to create a picoprocess and loads the EnclaveBootstrap (the bootstrap).
2) ***Shield module bootstrap:*** The bootstrap retrieves the shield module (the shield) from the testing database layer (the database) and bootstraps it.

3) ***Test case retrieval:*** The shield retrieves the desired VM bundle from the database, verifies its correctness and decrypts it. The shield loads the LibOS and VM image into the enclave.
4) ***Test case execution:*** The VM is started and is executed inside the enclave communication with it are performed in the same manner as with other components running inside an enclave.

4.4.3 Revised design

In this section we will present the revised design in accordance to the two design iterations we performed above. The TaaS solution is deployed as public cloud using Xen VMs on physical machines which contain a SGX processor. Each host will run Window 8.1 OS with a custom SGX driver. The solution's deployment is detailed in figure 29 and the architecture is detailed in figure 30.

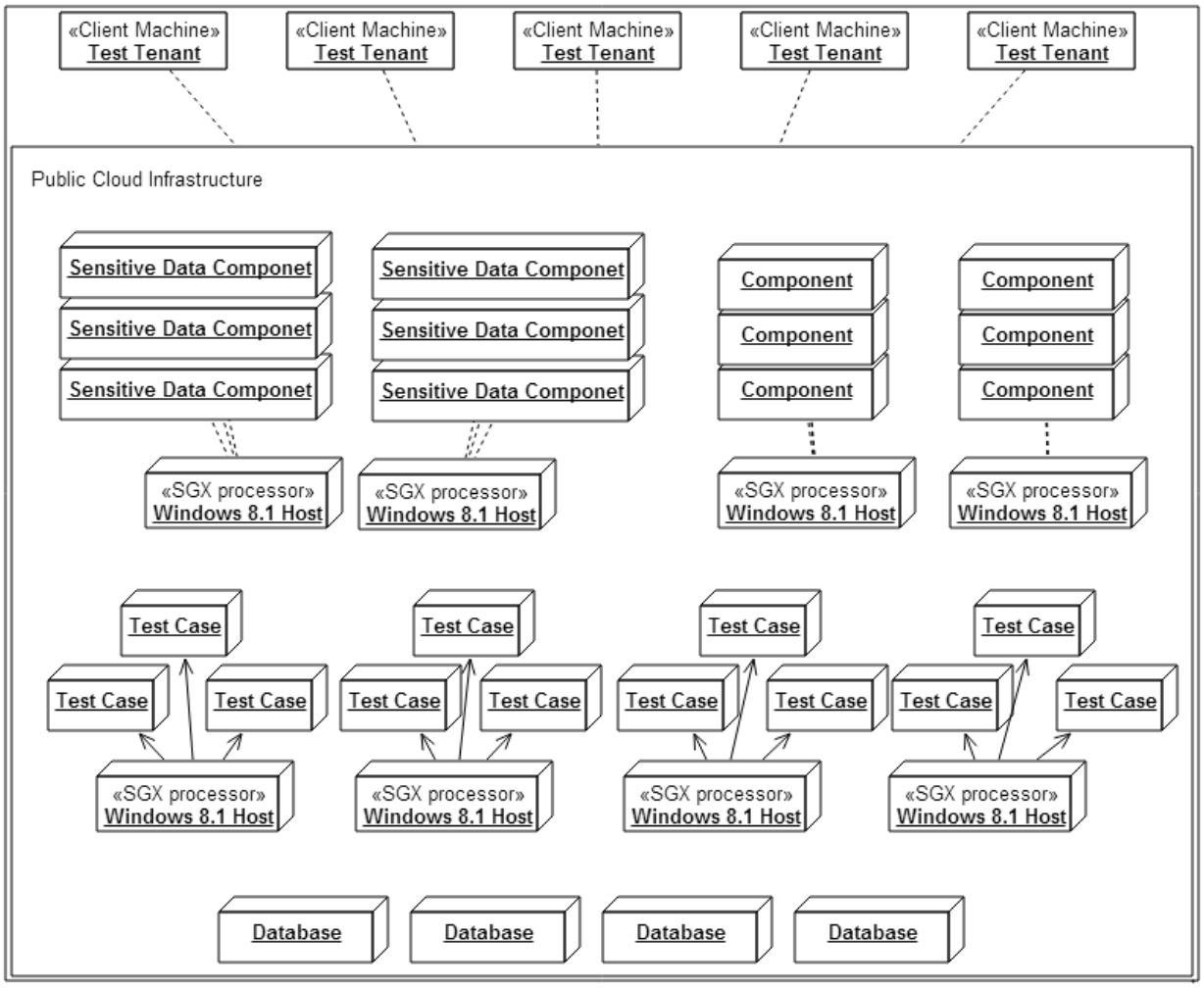

Figure 29 TaaS solution deployment

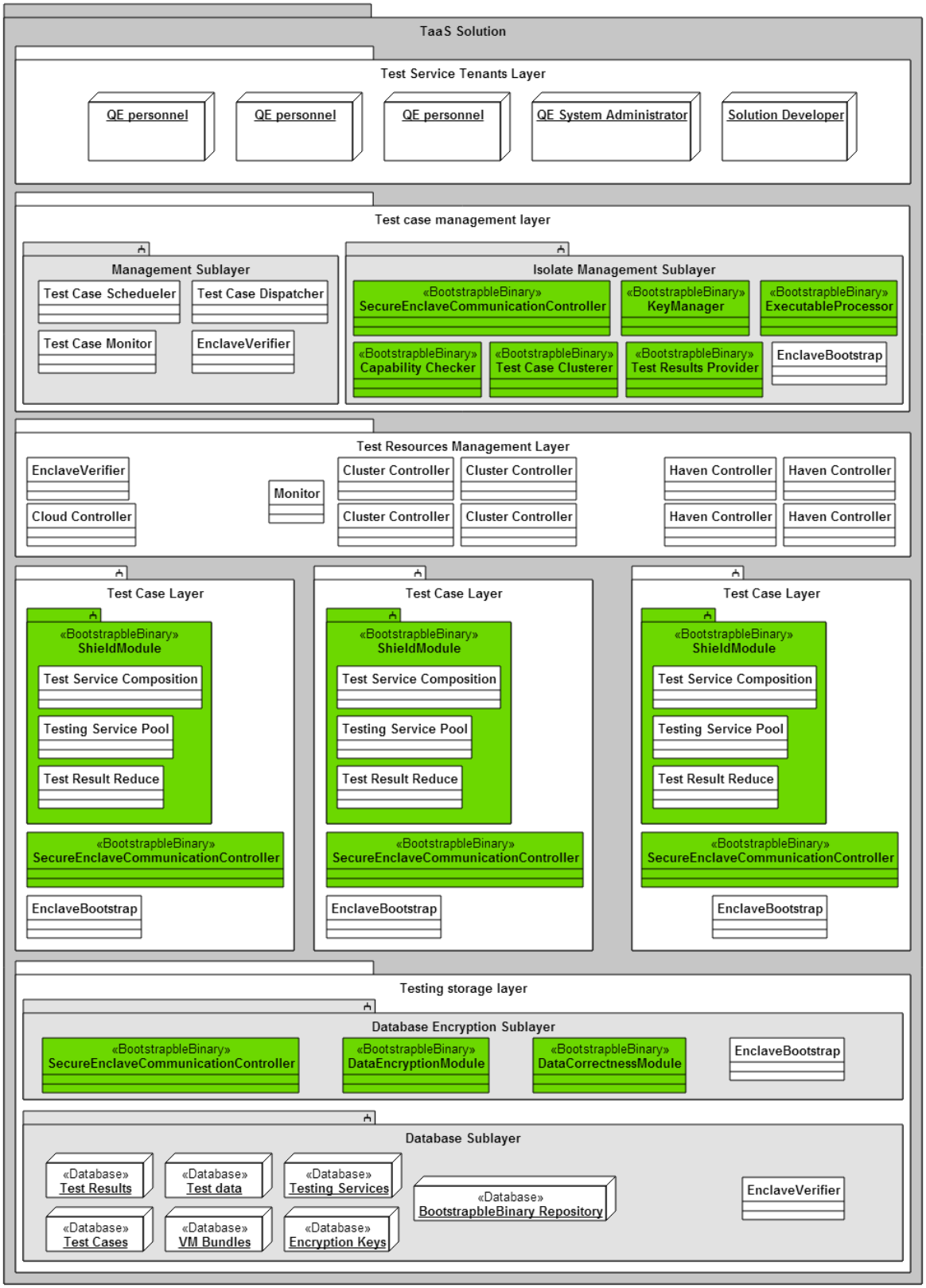

Figure 30 Revised TaaS Solution

5 Discussion

In this section we will discuss the design we proposed in section 4.4, we will show how it satisfies the functional and security requirements raised in section 4.2. We will then discuss the limitations of this design and what future work can be done in this regard.

5.1 Functional analysis

In this section we will review the functional requirements we have defined in section 4.2 and show how they are satisfied by the design proposed in section 4.4.

Looking at the revised design in section 4.4.3 we can see that we have not altered the functional behavior of any of the layers. Components that need protection are executed inside an enclave, with API calls being replaces by a communication mechanism between protected components and non-protected components. This communication mechanism allows various components to interact with each and continue operations as before.

Since the behavior of the solution remained the same we can show how the functional requirements are fulfilled:

1) ***Requirement*:** The customer must be able to manage test cases and test data within TaaS solution.
   Implementation: This requirement is fulfilled by the test service tenant layer and the test case management layer.
2) ***Requirement*:** The customer must be able to execute tests using the test cases and test data provided to the TaaS solution.
   Implementation: This requirement is fulfilled by the test service tenant layer, the test case management layer and the test case layer.
3) ***Requirement*:** The customer must be able to access the test results of the tests executed against the test data provided to the TaaS solution.
   Implementation: This requirement is fulfilled by the test service tenant layer, the test resources management layer and the testing storage layer.
4) ***Requirement*:** The TaaS solution provides a scalable test environment which would change according to the test suites needs.

Implementation: This requirement is fulfilled by the test case management layer and the test resources management layer.

5) ***Requirement*:** The TaaS solution must be able to simulate various environments in order to test various aspects of the application.
   Implementation: This requirement is fulfilled by the test case management layer, the test resources management layer and the test case layer.
6) ***Requirement*:** The TaaS solution must provide a contracting and billing service.
   Implementation: This requirement is implemented by the test resources management layer.

By augmenting the Drawbridge ABI to sealing and unsealing application state upon VM suspension and resume VM migration functionality is maintained. As mentioned in section 2.2 and in [12] the ability migrated VMs is an essential part of the test case clustering and execution capabilities. This allows the TaaS solution to provide customers with scalable and elastic testing environment.

In regards to the performance of the solution, according to [45] code executed inside an enclave suffers from a performance degradation of between 17%-23%. While applications executed inside a Haven execution environment [31] has suffered a performance degradation of 31%-54%, due to the added isolation and verification provided by the shield module and decryption of the VM bundles. We believe that this performance degradation is acceptable considering the security capabilities provided by this solution.

5.2 Security analysis

In this section we will review the security requirements we have defined in section 4.2 and show how they are fulfilled by the design proposed in section 4.4. Our solution is based on dynamic trust extension from SGX whose capabilities are detailed in section 3.3.6 and [28].

If we observer the solution architecture presented in sections 2.3 and 4.4.1 we can see that the following attack vectors exist:

- Attackers can target the data while it is active and attempt to access or modify the data while it being used by different components.

- Attackers can target the data while it is in transit between different components. They can then attempt to access or modify the data while it is transmitted.
- Attackers can target the data while it is at rest and attempt to access or modify it while it is stored within the database.

We will now show how data is protected in accordance to the attack vectors:

- ***Active data:*** As shown in section 4.4.2.1 when customer data is processed it is always done from inside an enclave. The bootstrapping process uses validation to ensure that a component's binaries have not been tampered with by an attacker in order to leak data from inside the enclave. The same bootstrapping mechanism is used in order to validate and load VM bundles into the Haven execution environment ensuring that test case execution environment are unaltered. Only after passing validation is a component or a VM bundle loaded into an enclave.

  We ensure that unencrypted customer data is only provided to components which are executed inside an enclave which is part of the CSP's infrastructure. This is done using the SecureEnclaveCommunicationController which is responsible for attestation and enclave communication with the outside world. The SecureEnclaveCommunicationController uses local attestation and remote attestation in order to verify that components are running inside an enclave which is part of the CSP's infrastructure.

  As stated in section 3.3.6 and [28] SGX provides memory protection from access and modification by entities that operate from out of an enclave. This includes: software attacks, hardware attacks such as memory probes, other enclaves running on the same host. Hence all customer data is safe while inside an enclave.

  Our component validation functionality relies on strong symmetric encryption and true random number generation which is used to generate a strong symmetric encryption key in accordance to [29]. As stated in section 3.3.6 SGX provides true random number generation, hence components can be encrypted and verified.

Our enclave verification functionality relies on an enclave's ability to verify the identity of other enclaves. The attestation mechanism presented in section 3.3.6 and SGX documentation is used for this purpose, hence enclave identities can verified. We have shown that only components which that are executed inside an enclave and are located inside the CSP's infrastructure have access to unencrypted customer data.

As we have shown customer data cannot be accessed or modified while it is an enclave. Hence customer data is secure while it is active due to the security capabilities which are provided by SGX.

- ***Data in transit:*** As shown in section 4.4.2 data is transferred between the application's components via the SecureEnclaveCommunicationController. We use TLS [43] which is a communication protocol which provides security for network communication. Utilizing TLS ensures that communication between two components cannot be eavesdropped or modified by unauthorized entities [43].

  The establishment of communication channels between enclaves and other entities is done by the SecureEnclaveCommunicationController attestation has been performed attestation. As the TLS protocol relies on a private/public key pair and random number generation for secure communication, the SecureEnclaveCommunicationController uses the pair which is provided by the enclave it is executed in and the *RDRAND* command.

  Our ability to create a secure communication channel relies on a strong public/private encryption key pair and generating random numbers. As stated in section 3.3.6 SGX provides strong private/public encryption key pair and true number generation, hence communication is secure.

  As we have shown customer data cannot be accessed or modified while it in transit due the security capabilities which are provided by SGX. Hence it is secure while in transit.

- ***Data at rest:*** As shown in section 4.4.2.2 customer data is stored in an encrypted form within the testing storage layer. Unencrypted data is transferred between enclaves and external entities via the SecureEnclaveCommunicationController using TLS.

  The database encryption sublayer (which is executed inside an enclave) is responsible for encryption of data before it is stored within the unprotected database and data validation and decryption before it is decrypted and retrieved by the various components (who are executed inside an enclave).

  The Blowfish symmetric encryption algorithm is used for encryption and the Shacham and Waters proof of storage algorithm is used for data correctness validation. The solution uses a KeyManager to generates, store and retrieve all of the symmetric encryption keys used for data encryption.

  Our symmetric encryption functionality relies on our ability to generate a strong symmetric encryption key through random number generation [29]. As stated in section 3.3.6 and [28] SGX provides true random number generation through the *RDRAND* command, hence data can be encrypted and decrypted using Blowfish.

  Our data validation functionality relies on strong symmetric encryption and strong random number generation which is used to generate a strong symmetric encryption key [29]. As stated in section 3.3.6 SGX provides true number generation through the *RDRAND* command, hence data can be encoded and verified.

  As we have shown while customer data can be read by an attacker when it is at rest the attacker is not able to use it. Any modification will be detected using the verification algorithm. These capabilities extend for the security capabilities provided by SGX. Hence data is secure while it is at rest.

We have shown that our TaaS solution provides protection to customer data. It cannot be accessed or modified by unauthorized entities due to utilization of the security capabilities which

are provides by SGX. Hence the security requirements are fulfilled. Hence the security requirements are fulfilled.

5.3 Limitations

In this section we will discuss what the current limitations of our research are.

Unfinished SDLC

As we discussed in section 3.3.6, at the time of writing no official processor support of SGX exists and emulator has only been supplied to selected partners such as Microsoft [31] [32]. We were not able to implement the core security functionality of our solution since we do not have access to a host which contains a SGX enabled processor or an SGX emulator. However our work is based on actual solutions and principals presented in the works we have cited.

Because our work is theoretical in nature, we are not able to run security and performance tests on our application. However as we have shown in sections 5.1 and 5.2 our solution fulfils the requirements of a secure TaaS solution.

Lack of time and manpower

As discussed in section 2.2 a TaaS solution is a cloud application which provides customer with various capabilities such as various customer interfaces (IDE plugins, web user interface, web services interfaces), various testing capabilities across different platforms, environment provisioning, billing services etc. Due to budgetary, man power and most important time constraints, it is not feasible to present a working TaaS solution within the scope of this work.

Access to development resources

As we discussed in section 3.3.6 Intel doesn't not offer any SGX support for any of it processors. It does however offer a SGX emulator to selected partners such as Microsoft [31] [32]. We have attempted to gain access to the SGX emulator by contacting Intel; however such attempts were not fruitful. Because of these reasons we were restricted to relying on Intel documentation and prototypes in order to understand how SGX operates.

Limited native VM management support

As we discussed in section 3.3.6 Haven does not support VM migration, this is problematic as VM migration is an essential part of a cloud solution's capabilities. We have suggested how this issue can be remedied, but since we have no access to the Haven's and LibOS code base we cannot perform these changes ourselves. Our reasoning for utilizing Haven and Drawbridge is the fact that other competing containers do not support SGX, although that SGX is built with containers (Docker specifically) in mind [46].

Limited OS support

As discussed in section 4.4.2.1 we are utilizing Haven in order to execute unaltered applications inside an enclave while protecting them from attacks by a malicious host OS. Haven utilized components (LibOS and Drawbridge) which are built on top of the Windows 8 OS.

Our reliance on the Microsoft technological stack limits the testing capabilities offered by our TaaS solution, to Microsoft only testing scenarios. Furthermore, no commodity OS support SGX capabilities as both the solutions presented in [31] and [32] use a custom SGX driver. Our reasoning for utilizing Haven and Drawbridge is the fact that other competing containers do not support SGX functionality, although that SGX is built with containers in mind [46].

5.4 Future work

In this section we will discuss what future work can be performed on the basis of our work.

Solution implementation and testing

As discuss earlier, at the time of writing no official processor support of SGX exists and emulator has been supplied to selected partners such as Microsoft [31] [32]. A future Implementation of this work should complete the implementation phase of the SDLC once SGX processors or SGX emulators are more readily available.

Since we couldn't supply a working solution we were not able to run functional, security and performance tests on our application. A future implementation of this work should complete the testing phase of the SDLC and perform both functional, performance and security tests.

Wider OS support

As discussed earlier we are utilizing Drawbridge and Haven in order to allow unaltered applications to execute inside an enclave. It also protects these applications from malicious host OS and Iago attacks. Haven utilized components (LibOS and Drawbridge) which are built on top of the Windows 8 OS. Our reliance on the Microsoft technological stack limits the testing capabilities offered by our TaaS solution, to Microsoft only testing scenarios.

This issue can be remedied by running a Linux VM from inside the Haven environment or by extending and using Docker [47]. Docker is a Linux open-source container virtualization platform. It utilizes a lightweight container virtualization platform for application management and deployment securely within a secure isolated container. However due to the fact that SGX processor emulators are not currently available this cannot be done yet.

Furthermore, no commodity OS support SGX capabilities. This can be remedied by extending open source Linux OS to support SGX However due to the fact that SGX processor emulators are not currently available this cannot be done yet.

Increased native VM management support

As discussed in section 4.4.2.1 Haven does not support VM migration natively, this is problematic as VM migration is an essential part of a cloud solution capabilities. We have suggested how that can be remedied but since we have no access to the Haven's and LibOS code base we cannot perform these changes.

These issues can be remedied by extending the Docker code base and then using it instead of Drawbridge. However due to the fact that SGX is not yet publicly available and no emulators are publicly available either, we cannot execute Docker inside an enclave to verify that it would function correctly.

Conclusion

Cloud computing offers customers the ability to reduce costs in regards execution of computation tasks and resource utilization. Using cloud solutions allows customers to quickly and simply provision various resources which can be used to store data, provision whole platforms and run applications without the need to maintain a physical infrastructure.

A cloud solution provides customers with an opportunity to migrate their software testing infrastructure to a cloud environment. It provides customers with scalability, easy resource provisioning and simple test environment setup. When coupled with TaaS solutions that provide various testing capabilities TaaS can provide a customer's QE personnel with services which would simplify the software testing process.

However these advantages are outweighed by the risks which are inherent to cloud services: by utilizing the CSP's resources customer trusts the CSP with full access to their data.

This is especially true in the case of TaaS solution. The advantages to executing tests using the virtually limitless resources provided by a cloud solution are outweighed by the risk of customer data leak. Any leak of customer data can cause great harm to a customer's reputation and business.

In order to address this issue we have presented a novel design for an end to end TaaS solution deployed in untrusted cloud environments. We utilize the security capabilities which are provided by Intel's SGX in order to provide strong security guarantees towards confidentiality and integrity of customer data.

Our solution protects customer data at all levels: at rest, in transit and while active under a strict threat model. We show that our design can be implemented using commodity hardware and commodity software with minimal modification.

We believe that our solution can be utilized, not only for TaaS solutions but for all cloud solutions who wish to perform computational tasks under the same strict threat model.

Bibliography

[1] P. Mell and T. Granface, "The NIST Definition of Cloud computing," National Institute of Standards and Technology, 2011.

[2] X. B. a. W.-T. T. Jerry Gao, "Cloud Testing- Issues, Challenges, Needs and Practice," *Software Engineering: An International Journal,* 2011.

[3] HP, "Testing and Quality Assurance Services," HP, 18 9 2014. [Online]. Available: http://www8.hp.com/us/en/business-services/it-services.html?compURI=1078997.

[4] Oracle, "Oracle Testing as a Service (TaaS)," Oracle, 9 September 2014. [Online]. Available: http://www.oracle.com/technetwork/oem/cloud-mgmt/testing-as-a-service--1905801.html. [Accessed 9 September 2014].

[5] J. H. Y. G. Siqin Chen, "Static Testing as a Service on Cloud," in *27th International Conference on Advanced Information Networking and Applications Workshops*, 2013.

[6] S. K. M. K. Changsup Keum, "Architecture based testing of service-oriented applications in distributed systems," *Information and Software Technology,* no. 55, 2013.

[7] W.-T. T. R. P. X. B. a. T. U. Jerry Gao, "Mobile Testing-As-A-Service (MTaaS) – Infrastructures, Issues, Solutions and Needs," in *IEEE 15th International Symposium on High-Assurance Systems Engineering*, 2014.

[8] F. Shull, "A Brave New World of Testing?," *IEEE Software,* pp. 5-7, 2012.

[9] O. T. K. S. Leah Muthoni Riungu, "Research Issues for Software Testing in the Cloud," in *2nd IEEE International Conference on Cloud Computing Technology and Science*, 2010.

[10] N. M. J. S. M. D. S. Juan Caballer, "Binary Code Extraction and Interface Identification for security applications," Electrical Engineering and Computer Sciences University of California at Berkeley, 2009.

[11] C. Group, "Cloud billing: The missing link for for cloud providers," 2010. [Online]. Available: http://www.cgi.com/files/white-papers/cloud-billing-white-paper.pdf.

[12] W.-T. T. X. C. L. L. Y. Z. L. T. W. Z. Lian Yu, "Testing as Service over Cloud," in *Fifth IEEE Internation Symposium on Service Oriented System Engineering*, 2010.

[13] O. T. K. S. Leah Riungu-Kalliosaari, "Testing in cloud: exploring the practice," *IEEE Software,* pp. 46-51, 2012.

[14] V. S. Subashini n, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications,* 2010.

[15] M. Spies, "A software assurance evidence approach to cloud security," in *22nd International Workshop on Database and Expert Systems Applications*, 2011.

[16] A. C. E. M. Anderson Morais, "A Model-Based Attack Injection Approach for Security Validation," in *SIN'11*, 2011.

[17] V. Getov, "Security as a Service in Smart Clouds – Opportunities and Concerns," in *IEEE*

36th International Conference on Computer Software and Applications, 2012.

[18] N. S. J. L. D. P. V. C. a. U. V. Massimiliano Rak, "Security as a Service Using an SLA-based Approach via SPECS," in *IEEE International Conference on Cloud Computing Technology and Science*, 2013.

[19] J. S. D. P.-B. T. Z. G. T. a. R. B. L. Pramod Jamkhedkar, "A Framework for Realizing Security on Demand in Cloud Computing," in *IEEE International Conference on Cloud Computing Technology and Science*, 2013 .

[20] L. L. J. X. R. H. N. A. Y. Z. Hussain Al-Aqrabi, "Investigation of IT Security and Compliance Challenges in Security-as-a-Service for Cloud Computing," in *IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2012.

[21] K. L. SENY KAMARA, "CRYPTOGRAPHIC CLOUD STORAGE," in *14th international conference on Financial cryptograpy and data security*, 2010.

[22] T. C. Group, "Development of Trusted Computing Standards," 2 24 2015. [Online]. Available: http://www.trustedcomputinggroup.org/trusted_computing/standards_development.

[23] C. M. S. C. a. Ramya Jayaram Masti, "An Architecture for Concurrent Execution of Secure Environments in Clouds," in *CCSW: The ACM Cloud Computing Security Workshop* , Berlin, 2013.

[24] A. Piltzecker, The Real MCTS/MCITP Exam 70-647 Prep Kit, 1st Edition, Syngress, 2008.

[25] S. G. Jordi Cucurull, "Virtual TPM for a secure cloud: fallacy or reality?," in *The Spanish Meeting on Cryptology and Information Security (RECSI)* , 2014.

[26] I. A. A. B. C. R. H. S. V. S. a. U. S. Frank McKeen, "Innovative Instructions and Software Model for Isolated Execution," in *2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[27] Intel, "Intel® SGX for Dummies (Intel® SGX Design Objectives)," Intel, 20 September 2013. [Online]. Available: https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx. [Accessed 7 March 2015].

[28] S. G. S. P. J. V. R. S. Ittai Anati, "Innovative Technology for CPU Based Attestation and Sealing," in *The Second Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[29] A. R. Elaine Barker, "Recommendation for Cryptographic Key Generation," NIST, 2012.

[30] D. B. Young, "Foundations of Computer Security: Symmetric vs. Asymmetric Encryption," [Online]. Available: https://www.cs.utexas.edu/users/byoung/cs361/lecture44.pdf.

[31] Microsoft Research, "Shielding applications from an untrusted cloud with Haven," in *11th USENIX Symposium on Operating Systems Design and Implementation.*, 2014.

[32] M. C. C. F. C. G. M. P. M.-R. a. M. R. Felix Schuster, "VC3: Trustworthy Data Analytics in the Cloud," in *36th IEEE Symposium on 36th IEEE Symposium on*, 2013.

[33] O. o. I. S. 1, "Selecting a development approach," 27 March 2008. [Online]. Available: http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf.

[34] C. Gentry, "A fully homomorphic encryption scheme," 2009.

[35] Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," 2010.

[36] D. B. C. Kurt Rohloff, "A Scalable Implementation of Fully Homomorphic Encryption Built on NTRU," in *2nd Workshop on Applied Homomorphic Cryptography and Encrypted Computing*, 2014.

[37] S. H. Craig Gentry, "Implementing Gentry's Fully-Homomorphic Encryption Scheme," in *EUROCRYPT* , 2010.

[38] S. Halevi, "An Implementation of homomorphic encryption," 13 April 2013. [Online]. Available: https://github.com/shaih/HElib.

[39] J. Rutkowska, "Thoughts on Intel's upcoming Software Guard Extensions (Part 1)," 30 August 2013. [Online]. Available: http://blog.invisiblethings.org/2013/08/30/thoughts-on-intels-upcoming-software.html. [Accessed 7 March 2015].

[40] V. S. J. A. R. M. Ranjeet Masram, "Analysis and comparison of symmetric key cryptographic algorithms based on various file feature," *International Journal of Network Security & Its Applications,* 2014.

[41] C. B. J. G. N. Aiiad Albeshri, "A Security Architecture for Cloud Storage Combining Proofs of Retrievability and Fairness," in *CLOUD COMPUTING 2012 : The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, 2012.

[42] D. W. M. B. T. R. Michael Coates, "Transport Layer Protection Cheat Sheet," 10 3 2015. [Online]. Available: https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet#SSL_vs._TLS. [Accessed 15 3 2015].

[43] Network Working Group, "The Transport Layer Security (TLS) Protocol Version 1.2," August 2008. [Online]. Available: http://tools.ietf.org/html/rfc5246?as_url_id=AAAAAAVBehpzRqATU5xWpMSTPjTY4oV6aOnai43OyHdsdcjqdSlYu0y-i_wtuyMcDhdfR_le_fBCnWW1xu50YwXZ7oot.

[44] D. L. P. F. L. G. R. L. B. B. R. O. G. C. H. Andrew Baumann, "Composing OS Extensions Safely and Efficiently with Bascule," in *8th ACM European Conference on Computer*

Systems, 2013.

[45] R. L. P. P. C. R. V. P. J. d. C. Matthew Hoekstra, "Using innovative instructions to create trustworthy software solutions," in *2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[46] I. C. Nicholas Weaver – Principal Architect, "Bare-metal, Docker Containers, and Virtualization: The Growing Choices for Cloud Applications," in *Intel Developer forum*, 2014.

[47] "Docker Build, Ship and Run Any App, Anywhere," Docker, [Online]. Available: https://www.docker.com/.

[48] D. Booth, "Web Services Architecture," The World Wide Web Consortium, 2004. [Online]. Available: http://www.w3.org/TR/ws-arch/#service_oriented_architecture.

[49] S. T. Brianna Floss, "Software Testing as a Service: An Academic Research Perspective," in *IEEE Seventh International Symposium on Service-Oriented System Engineering*, 2013.

[50] F. M. Wang Jun, "Software Testing Based on Cloud Computing," in *2011 International Conference on Internet Computing and Information Services*, 2011.

[51] J. X. B. W.-T. T. Gao, "Testing as a Service (TaaS) on Clouds," in *IEEE Seventh International Symposium on Service-Oriented System Engineering*, 2013.

[52] Oracle, " Oracle Application Testing Suite," Oracle, 18 September 2014. [Online]. Available: http://www.oracle.com/technetwork/oem/app-test/etest-101273.html. [Accessed 18 September 2014].

[53] HP, "Test automation minus the investment, minus the risk," HP, 18 9 2014. [Online]. Available: http://h30507.www3.hp.com/hpblogs/attachments/hpblogs/EnterpriseServicesBlog/76/1/HP.

[54] S. D. C. d. V. Pierangela Samarati, "Data protection in outsourcing scenarios: issues and directions," in *5th ACM Symposium on Information, Computer and Communications Security*, 2010.

[55] D. Davidowicz, "Domain Name System (DNS) Security," 1999.

[56] M. J. G. Jeremy Daniel Wendt, "Trusted Computing Technologies, Intel Trusted Execution Technology," Sandia National Laboratories, 2011.

[57] i. A. G. Narender Tyag, "Comparative Analysis of Symmetric Key Encryption Algorithms," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 4, no. 8, pp. 348-354, 2014.