Master's Thesis

Trusted Computation based Ecosystem for Blockchain and Beyond

submitted by

Mujtaba Idrees

Technische Universität Dresden

Faculty of Computer Science
Institute for Systems Architecture
Chair of Systems Engineering

Supervisors:
Dr.-Ing. Do Le Quoc
Dr.-Ing Ivan Gudymenko

Reviewers:
Prof. Dr. Christof Fetzer
Dr. André Martin

Professor:
Prof. Dr. Christof Fetzer

Submitted December 14, 2020

Task description - Master Thesis

Name, Vorname:	Idrees, Mujtaba
Studiengang:	Distributed Systems Engineering
Matrikelnummer:	4807174
Thema:	***Trusted computation based ecosystem for blockchain and beyond***

Blockchain provides computational trust by replicating the ledger state across the nodes. Most blockchain technologies of today support smart contracts which are executed by miners in the network. In doing so the contract between two parties becomes available for every node in the network. Essentially, we trade privacy for trust. For a better blockchain adaptability and integration in real-world systems the blockchain technology of future must be privacy preserving without trading off the trust factor.

Meanwhile a lot of work is being done in the field of trusted computing. By means of trusted computing we can ensure that correct program is being executed and execution of a particular workload is not visible to other processes and even the root admin, thus mitigating the possibility of meltdown or side channel attack. Pertaining to the privacy benefits of trusted computing it makes sense to leverage its properties to make the future blockchains more secure and privacy preserving. Moreover, if we somehow off-load the heavy computational work from blockchain to separate worker nodes it would make the blockchains lightweight and more scalable.

Hyperledger foundation has an ongoing project named "Hyperledger Avalon" that is trying to solve the same problem. It aims to enable creation of a network of dedicated trusted compute workers which could serve both blockchains and non-blockchain based clients. The Avalon project is still in its incubation phase and they have published an opensource architecture of the product and a demo application as a proof of concept with minimum functionality i.e. only a singleton worker node of Intel SGX based trusted execution environment. The key management of trusted compute workers and their implementation in scalable and fault tolerant clusters is still an issue for Avalon developers.

There are multiple platforms that support application deployment upon Intel SGX based trusted compute workers and one of them is "SCONE". It enables trusted execution of complex applications in docker based containers on Intel SGX hardware. It has inherent support for Kubernetes based clusters and has an efficient mechanism of key management and attestation. Hence SCONE platform can be reused and plugged inside Hyperledger Avalon's current design to make it more scalable and fault tolerant.

This thesis proposes a ***Trusted Computation based Ecosystem for Blockchain and Beyond (TCEBB)*** – An ecosystem that would enable thin and secure future blockchains by off-loading the heavy computations to outside trusted workers. At the same time, it can also be used as privacy preserving function as a service for end users. The main idea of the thesis would be to fork the opensource code of Avalon at current point or some stable release that supports blockchain integration and basic registry service for trusted compute workers and then merge it with SCONE based clusters that it supports out-of-box.

The thesis would have the following high-level tasks:

- Fork Avalon to have SCONE workers
- Evaluating how to fit SCONE out-of-box into Avalon ecosystem
- Use SCONE with Kubernetes to support worker pools in Avalon
- Evaluating key management techniques of SCONE & Avalon and figure out how to use SCONE's key management out-of-box
- Support fault tolerance in scone based trusted workers
- Compare Avalon's native workers with SCONE based worker pools (If available)

This thesis would be done in collaboration with T-Systems MMS and is a possible service for German Blockchain Ecosystem (GBE). It will be written in English.

Betreuer:	Dr.-Ing Do Le Quoc (von TU-Dresden)
Zweitbetreuer:	Dr.-Ing Ivan Gudymenko (von T-Systems MMS)
Zweitgutachter:	Dr. Andre Martin
Verantwortlicher Hochschullehrer:	Prof. Christof Fetzer
Institut und Lehrstuhl:	Systemarchitektur, Systems Engineering
Beginn am:	01.07.2020
Einzureichen am:	31.12.2020


Student


Betreuer


Zweitbetreuer


Zweitgutachter


Verantwortlicher HSL

Confirmation

I confirm that I independently prepared this thesis with the title *Trusted Computation based Ecosystem for Blockchain and Beyond* and that I used only the references and auxiliary means indicated in the thesis.

Dresden, December 14, 2020

Mujtaba Idrees

Abstract

Blockchain adoption has improved significantly over the last few years and many blockchains customized for various use cases have emerged. The blockchain technology stack has also been improved quite a lot since Bitcoin, the first practical blockchain implementation. However, due to their design, the blockchains of current era still pose some challenges like privacy, scalability and interoperability.

This thesis presents a neoteric approach to solve the above-mentioned problems of existing blockchains, already running in production environments. A trusted computation-based ecosystem is proposed; which enables confidential execution of custom workloads, enhances blockchain scalability and enables interoperability of diverged blockchain networks.

Legacy applications can be executed in SCONE based secure containers over an ecosystem of trusted compute workers provided by Hyperledger Avalon. While the presented ecosystem solves the problems of current blockchains, it can also be leveraged by non-blockchain based clients, enabling wide range of use cases beyond blockchains.

Contents

1 Introduction

A function is a set of instructions, executed step by step in a computer, to perform a task. Multiple functions constitute to make applications. We use thousands of applications in our daily life. They run on our laptop, on our smart phone, or on the servers running as web services. Our manual work from daily lives have moved to such applications which are less prone to human error, provide fast processing, better resource management and automation of tasks. But it was not always like we see them today. Initially, the computation resources were scarce and only complex workloads used to run on private mainframes or on personal computers. As the technology advanced, it took over the world and then came the cloud revolution.

In cloud revolution, majority of computational tasks and data moved to public and private clouds, managed by centralized authorities. Private cloud allowed the service providers to flexibly manage user's data and resources at their premise. Most successful companies of this century like Google, Facebook, Amazon were made possible due to private clouds. Public cloud on the other hand, allowed ease of delivery to application providers, who could focus on the application development without worrying about the infrastructure management. They could simply rent the cloud infrastructure and manage load on the principal of 'pay per use'.

The cloud solutions of today are centralized in nature, so they are single hotspot for security attacks and more susceptible for leaks like Cambridge Analytica [Bol18], Equifax data breach [Tho19] etc. Hundreds of such breaches on public and private cloud have occurred in the last two decades [BA17]. Besides the data breaches from outside entities, Edward Snowden's revelations [Alh+15] show us that a breach can also occur from inside the cloud provider and the data and processing is susceptible to exploitation by root admin and side channel attacks [PS10]. Even if we trust the intentions of the cloud provider, we cannot establish a computational trust and privacy in traditional centralized cloud. One way to establish trust is decentralization which leads us to the blockchain revolution.

Blockchains [KVC19] provide computational trust by replicating the state of the ledger, distributed amongst nodes in a cluster. Due to implicit trust provided by design in blockchains, many applications of today such as cryptocurrency [Nak] are made possible. Modern blockchains allow the users to program application specific functions termed as 'smart contracts' [AAM19] which can be executed in a decentralized way, such that the trust on the execution can be established. While they provide computational trust, the traditional blockchains of today do not support privacy by design. This is due to the fact that blockchains establish trust in smart contracts by executing them across multiple nodes in the network. Essentially, we trade-off privacy for trust.

The execution of a smart contract is bound by a fee, that is paid to the miners by the user who

initiates the execution of the transaction. The amount of fee usually depends upon the complexity of the smart contract workload. This way developers are encouraged to write light weight smart contracts and keep the blockchains offloaded with bigger workloads. Hence, the trust of blockchain cannot be leveraged by all types of use cases applicable in real world. Only a limited set of use cases can be adapted to blockchain.

One way to bridge the trusted world of blockchains with the untrusted world, is using blockchain oracles. Blockchain oracle is an entity which feeds the external data such as weather updates, currency conversion rate, result of mathematical modeling etc. from the untrusted off-chain source into the blockchain after its verification. Using oracles also enables us to offload some processing from the blockchains. In order to trust the data, trust on the oracle needs to be established. There are some solutions such as chainlink [EJN], which establishes the trust on oracles, using decentralized proof of stake [Ngu+19] approach.

We discussed that computational trust and privacy cannot be established for traditional cloud, and privacy cannot be established for traditional blockchains. For better adaptability of blockchains and their integration in real-world systems, the blockchain technology of future must provide trust and privacy both. Similarly, for end-to-end protection of data and execution in cloud, the cloud of future must preserve integrity and confidentiality of execution.

Integrity and confidentiality of an execution can be established by the means of trusted computing. Trusted computing [She+10] constitutes of three categories: Zero Knowledge Proofs (ZKP) [Has19], Multi Party Compute (MPC) [Bol18]3 or hardware-based Trusted Execution Environments (TEE) [SAB]. Zero knowledge proof is a cryptographic protocol, using which one party can prove a certain information to another party without revealing any of its details. For example, a person can prove that his age is above 18 if he owns an identity card, without revealing the exact age. Multi party compute is another way to execute a transaction securely be dividing its execution between multiple unrelated parties such that one party cannot make sense of the execution without seeing other's data. Trusted execution environment (TEE) is a reserved area in a hardware in which only trusted execution takes place. It is totally isolated environment inside the CPU, in which no other process can intervene. Only trusted code can be loaded in the TEE and sanctity of hardware and code is attested before execution takes place to make sure the setup has not been tampered with. The interactions of the application with outside the TEE are totally encrypted with the private keys residing inside the TEE. Hence, the execution is resistant to exploitation by root admin and side channel attacks.

Currently, a lot of work is being done in the field of trusted computing, and practical solutions such as TEEs using Intel SGX [Fuk19] are available which can be leveraged to make clouds and blockchains more secure and privacy preserving. Particularly, blockchains can be made more efficient by offloading their heavy computations into TEEs and allowing them to interact with outside world using attested oracles supported by TEEs.

In this thesis, a ***Trusted computation-based ecosystem for blockchains and beyond (TCEBB)*** is presented. An ecosystem that enables lightweight and secure future blockchains by allowing them to run heavy workloads in trusted execution environments. At the same time, this ecosys-

tem would also provide a privacy preserving function as a service for cloud workloads. Using this ecosystem, real world computation intensive legacy applications, such as machine learning algorithms, data science workloads, Artificial Intelligence based processing etc. can be executed in trusted workers without code instrumentation which can be integrated with any blockchain or non-blockchain client.

The main idea of this thesis is inspired from opensource project "Hyperledger Avalon" [Tea], which tries to solve the problems stated above.

1.1 State of the art

Hyperledger Avalon is a project incubated at Hyperledger foundation that enables creation of an ecosystem of trusted compute workers which could serve both blockchain and non-blockchain based clients. It is an opensource project having its detailed roadmap and architecture published [Yar], however it is still in its pre-release version. Its basic infrastructure is developed, and some demos are available on GitHub [Avaa]. In the current stable release, Avalon supports Intel SDK and Graphene based trusted workers in singleton mode. However, it can be extended to support more types of trusted compute workers.

Most of the state-of-the-art solutions in the field of secure computation originates from Intel SGX. Intel SGX is a practical solution that constitutes of specified hardware. It allows the trusted execution of code in Intel based trusted enclaves. Due to newly added hardware instructions in Intel SGX, intel provides its own SDK also known as Intel SDK, which can be used to program the workload for running inside Intel SGX based trusted execution environment.

As Intel SGX introduces its own hardware instructions, legacy applications cannot run inside Intel SGX based secure enclaves without code instrumentation. However, there are multiple platforms that support running of legacy applications inside Intel SGX hardware which include Graphene [TPV] and Secure Container Environment (SCONE) [tea] [Arn+16]. SCONE enables trusted execution of legacy applications inside docker based containers in Intel SGX. It has inherent support for Docker Swarm and Kubernetes based clusters and has an efficient mechanism for code attestation and key management through its CAS [Tra+18].

1.2 Our contributions

The opensource code of Hyperledger Avalon is forked at pre-release version 0.6. The forked version of Hyperledger Avalon is made to work with SCONE based trusted worker pools so that legacy applications can be run as trusted workloads.

The main architecture of Hyperledger Avalon remains unchanged. SCONE worker manager and SCONE work order processors have been added as an addition to Avalon nodes. The worker managers have been extended to support pool of 'N' workers. SCONE's attestation mechanism using

configuration and attestation service (CAS) has been added to attest SCONE based workers and to provision them the secret keys, after they bootup. The forked system is backward compatible and can support other workers as well as SCONE in production environment.

Using this infrastructure, SCONE curated images of applications can be fetched from Dockerhub and be easily integrated with Avalon. In order to better present the use case some real-world applications have also been developed as examples.

1.3 Evaluation results

Evaluation of the system was done from three different aspects. (1) Performance comparison of SCONE with other TEEs (2) Performance comparison of different modes of SCONE (3) Performance benchmarking of scalability of SCONE. The results of experiments from these different aspects are discussed step by step.

1. **Performance comparison of SCONE with other TEEs**
   In these experiments we ran performance benchmarks on SCONE based workers, Intel SDK based workers and Graphene based workers running with Hyperledger Avalon. It was observed that in smaller workloads the performance of Intel SDK was better as compared to SCONE but for complex workloads the performance of SCONE and Intel SDK is similar in hardware mode. When compared with Graphene, SCONE had better performance for both small and large workloads. The performance of SCONE grows gradually as the complexity of workload increases. For the largest workload SCONE was found almost twice as performant as Graphene.

2. **Performance comparison of different SCONE modes**
   In these experiments we ran performance benchmarks on SCONE simulation mode, hardware mode and hardware mode with filesystem protection enabled by CAS. These modes of SCONE were compared with the native mode. It was observed that for smaller workloads the performance of SCONE simulation mode is better than native execution, but it degrades gradually as the complexity of workload increases. For largest workload, the latency of simulation mode is slightly more than native mode.

   While simulation modes remain close to native execution, the maximum overhead of 2x over native execution was observed in hardware mode.

   The highest latency was observed for hardware mode with file system encryption enabled by CAS. In this mode, each change in file system leads to re-encryption of file and re-calculation of tag responsible for integrity check. This mode is crucial for end-to-end encryption and secure execution but has the highest overhead.

3. **Performance benchmarking of scalability of SCONE**
   In these experiments, the impacts of increasing SCONE workers in a pool were analysed. In

Avalon, the requests are processed sequentially by each worker. If there are multiple workorders to be processed by a worker, they are put in a wait queue. To manage the increasing load of requests, multiple workers can be added and the load can be balanced such that there is less wait time for any request. We analysed the impacts of running 1 to 5 workers in a pool on same hardware and it was observed that adding each worker increases the response time of the request due to sharing of hardware resources. However, overall speedup of 2.6x was achieved for the largest workload with 5 parallel workers running on same hardware.

1.4 Outline of the thesis

- Chapter 2 discusses the researches and related work done in the field of our study
- Chapter 3 provides necessary technical knowledge required to understand the concepts presented in this thesis
- Chapter 4 discusses the high-level design and architecture of the system presented in this thesis
- Chapter 5 discusses the implementation details and some example usecases developed in this work
- Chapter 6 discusses the evaluation results of the system
- Chapter 7 discusses the design improvements for future and further research topics that stem from this work

2 Related Work

In this section, we will be discussing the related work done in the field of trusted computing. We will be discussing two areas in this domain: (1) providing trusted computing as a service (2) using trusted computing to offload intensive blockchain computations

2.1 Towards Trusted Cloud Computing

This paper [SGR09] discusses the architecture of Trusted Cloud Computing Platform (TCCP) which enables Infrastructure as a Service (IaaS) providers to provide guarantees to their clients that their processing is being done in a confidentiality and integrity preserving manner. The service providers use TCCP to spin up the workloads of the clients in secure Virtual Machines. They do this by providing a closed-box execution environment using the mechanisms of Trusted Platform Modules (TPM) and secure booting.

Using TCCP, clients can attest IaaS providers and can make sure that the service is secure before they execute their applications. This way, the client can rest assured that their data would remain private and hence would not be misused.

TCCP has two components: Trusted Virtual Machine Monitor (TVVM) and a Trusted Coordinator (TC). TVVM enforces that no malicious sysadmin can tamper with data or processing inside the Virtual Machines of the clients. Whereas, TC manages the set of nodes of virtual machines that run the customer's applications.

This paper only discusses the architecture of such a system and doesn't have any implementation. Hyperledger Avalon is a complete functional prototype of a system that can run trusted nodes for clients using docker containers. Moreover, they also enable securely running offloaded workloads from blockchain. In our thesis contribution, we have further built upon Avalon and have added support for SCONE based workers which would give clients further flexibility to run secure workloads on Avalon without code instrumentation.

2.2 Challenges for Combining Smart Contracts with Trusted Computing

This paper [BC18] discusses the utility of offloading smart contract execution in trusted execution environments. It further discusses that by doing smart contract executions off-chain, the performance of blockchain can increase as doing such computations on-chain are costlier due to the cryptographic protocols used. It also states that by using TEE for smart contracts, it can also

increase interoperability between different types of blockchains.

However, it also states the challenges that arise when trying to offload the executions. These challenges include (1) inability to create complex workflows (2) challenges in attestation and key-management and (3) non-deterministic execution.

The first challenge is that in on-chain smart contract executions, complex workflows can be created by using the ability of one smart contract invoking another smart contract. Such workflows would be difficult to achieve with TEE based off-chain executions. It would need to be decided on a case-to-case basis whether using TEE for smart contracts is a feasible choice.

The second challenge is that attestation is expensive for smart contracts in Intel SGX and there is a need to create a light-weight attestation mechanism. Moreover, to ensure availability of smart contract TEE, multiple nodes for the same smart contract need to maintained. This would require key management which is another technical challenge. Moreover, key revocation would also need to be considered. The paper states that previously proposed systems do not provide satisfactory solutions and this is still an open concern. The third challenge is inability to distinguish between a TEE that is behaving incorrectly and a TEE that has non-deterministic executions.

As opposed to this paper, our thesis firstly supports both blockchain and non-blockchain based workloads. However, when it comes to blockchain based workflows, in our thesis, the first and third challenge remains as it is. However, the second challenge has been mitigated. We have used SCONE CAS which simplifies the IAS attestation process and is capable to do key management for us. CAS securely generates and maintains the keys for our trusted workers. If the mrenclave of the workers change, they would not be able to get the required secrets and keys necessary for proper functioning of the system.

2.3 ShadowEth

This paper [Yua+18] describes the problem of privacy in blockchains. In blockchain, data is replicated on all nodes in a network so it is not privacy-preserving by design. ShadowEth describes a system where execution of Ethereum smart contracts can be done in private Trusted Execution Environments and only puts the process of verification on the Blockchain.

Our thesis is similar to what ShadowEth is doing. However, it is not limited to Ethereum only. As Hyperledger Avalon is a complete ecosystem and we have further added the functionality of SCONE containers in it, therefore, other blockchains or workloads apart from Ethereum can also use it.

2.4 Enterprise Ethereum Alliance Off-Chain Trusted Compute Specification

Ethereum Enterprise Alliance (EEA) Off-Chain Trusted Compute Specification [BYZ] is set of guidelines created for off-chain trusted computation in Blockchain. All upcoming projects are expected to contribute to these goals of EEA for technical progress in this space. It basically has four goals:

1. Support of private blockchain transactions between distrusting parties
2. Disclosure of selected information to chosen parties (selective privacy)
3. Offload intensive computation from blockchain to trusted compute workers
4. Support of attested oracles (trusted data sources)

Our thesis follows these guidelines and the overall ecosystem of Avalon and SCONE that we have created are built upon the foundations laid by EEA.

2.5 OpenStack Trusted Compute Pools

OpenStack's trusted compute pools [Ope] allow sysadmins to create a pool of trusted nodes. The nodes use Intel's Trusted Execution Technology (TXT) and have an external stand-alone remote attestation server. Using the attestation server, it can be verified that nodes in the trusted pool are running with the correct state using the software stack's measurements.

This tool is an offering of open stack which customers can buy to manage their own clouds. Our thesis, on the other hand manages the creation of trusted pools and key management on its own. Users can use it as a service.

2.6 Amazon's NitroEnclaves

Nitro Enclaves [Ama] is Amazon's offering for customers that enables them to create isolated compute environments for handling sensitive customer's data. It uses Nitro Hypervisor Technology that is also used in Elastic Compute Cloud (EC2) for CPU and memory isolation.
This is an independent offering from amazon, whereas our thesis provides a complete ecosystem that is deployable on any cloud. Therefore, there is no cloud vendor lock-in. Moreover, we also provide a way to the customer to deploy their code without code instrumentation using SCONE.

2.7 Azure Confidential Computing

Microsoft Azure also has its own offering for confidential computing backed by Intel SGX [Azu]. It uses Open Enclave SDK for development. A user can either buy a plain Virtual Machine or a Confidential Compute Enabled virtual machine.

3 Technical Background

In this section, the background knowledge necessary to develop understanding of the topic and the technologies which are trivial to this work are discussed. We would explain various topics starting from Blockchain technology which is one of the main use cases of this thesis. Then we move to dockerized containers, upon which the secure containers stack of SCONE is based. The crux of this thesis is to enable secure execution of computation workloads in trusted execution environments which are explained in depth in this chapter. Various implementations that allow us to run trusted workloads in Intel SGX i.e. Intel SDK, Graphene and SCONE are also discussed in detail. In the end we would explain the design and implementation of the main topics of this thesis i.e. SCONE and Hyperledger Avalon.

3.1 Blockchain

3.1.1 Introduction

Blockchain is a decentralized ledger technology that can be used to keep store of records among multiple nodes in a cluster. A cluster of nodes in a typical blockchain network cannot be controlled by a single entity, hence the system is decentralized in nature. The data from blockchain is computationally validated by majority of unrelated nodes in a cluster of arbitrarily 'N' number of nodes. This core design of blockchain ensures that it is not controlled by a central authority and the system is overall 'trustless'. By trustless, we mean that there is no need to trust the source of data or provider, mathematical calculations and decentralized validation makes the system computationally trustworthy.

This design of blockchain ensures that the data has not been tampered with, as it does not have a single administrator that can update and delete the stored information. Due to its design, it exhibits the properties like immutability, irrevocability and non-repudiation. Such properties make it ideal solution for some important real-world use cases such as financial applications.

3.1.2 Design

On a technical level, blockchain can be explained as a chain of blocks, which keep the information about a set of transactions that have been made until the latest block as shown in Figure 3.1. The block headers keep the information of their previous block hash, its own hash, a list of transactions and time stamp. These attributes are used to trace a particular transaction in the blockchain network. Once a block is finalized and agreed by majority of stakeholders in the network, all the nodes in the blockchain save that block as the latest state of blockchain.

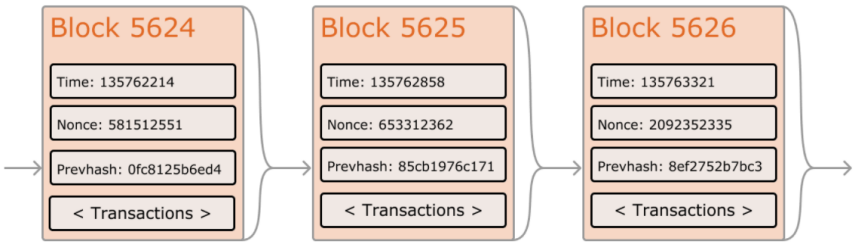

Figure 3.1 – Blocks chained together in a blockchain [But]

In order to have a consensus on the selection of a block, various mechanisms like proof of work (PoW) [Ger+16], proof of stake (PoS) [Ngu+19] and proof of authority (PoA) [De +17] etc. are available.

Blockchains on a higher level can be divided into two categories i.e. permissioned and permissionless. The permissioned blockchains further consist of consortial and private blockchain and permissionless blockchains refer to public blockchains. The details of these categories are discussed below.

Public Blockchains
Public blockchains are permission-less blockchain, in which everyone can run an opensource protocol on their machines and join the network as an equal node. The public blockchains operate in a peer to peer manner in a total decentralized way. They make an ideal solution for implementation of digital cryptocurrency. The most famous public blockchains are Bitcoin and Ethereum.

Private Blockchains
Private blockchains are distributed clusters of nodes running in a private intranet, storing and processing the data on the blockchain protocols. Private blockchains are not decentralized in nature, rather they are controlled by a centralized authority. Their main purpose is to provide scalable, distributed ledger. Usually lightweight proof of authority (PoA) protocol is used to create new blocks in private blockchains.

Consortial Blockchains
Consortial blockchains, like private blockchains are permissioned blockchains and are not open for public to join them. They represent a consortium of privileged parties which can be part of this network. The nodes are distributed in cluster spanning across various participant organization's intranets. They are particularly useful when mutually trusting parties want to leverage the properties of distributed ledger in a privacy preserving manner. The most famous consortial blockchains are Hyperledger Fabric, Hyperledger Indy etc.

3.1.3 Blockchain Workloads

Blockchains of today provide the ability for the users to write custom code pieces, which are executed as functions inside the blockchain ecosystem and the results of these executions become part of updated state of blockchain. Typically, these functions are executed at multiple nodes in

the network and a consensus is achieved on the state of results. In Ethereum and some other public blockchains these functions or workloads are called smart contracts, while in Hyperledger Fabric these workloads are called chaincodes.

3.1.4 Future

Blockchain technology is maturing by each passing day and in recent few years many new blockchains have surfaced, each having a different design and implementation specially curated to the targeted use case. There are also some drawbacks of traditional blockchain design which are open issues and many implementations are targeting to solve these problems. The current open problems of blockchains of today are (1) Lack of privacy in transaction execution (2) High transactional latency due to complicated consensus mechanisms that creates scalability issues (3) Interoperability between different blockchains (4) Integration of blockchains with outside world.

The newer blockchains implementations such as Zcash, Polkadot, Ethereum 2.0, Chainlink etc. are individually trying to solve these issues one way or another. In this thesis we are also aiming to solve the above-mentioned problems of traditional blockchains using a neoteric way i.e. using trusted execution environments. The workload from the blockchain is offloaded into trusted execution environments, which makes the blockchains lightweight and provides privacy in transactions execution. Besides this, the Hyperledger Avalon have implementations of blockchain connectors for various blockchains which can be leveraged to enable interoperability between different blockchains. The Avalon infrastructure can also be leveraged to create 'attested oracles' to integrate the blockchains with outside worlds data.

3.2 Docker Containers

Docker is a platform that allows packaging of applications in lightweight and deployable containers. All the dependencies of the application are packaged inside these containers, which provide a self-contained execution environment for an application. The applications running inside containers share the OS kernel resources but have isolated execution environments. From application's perspective, it runs inside an operating system on real hardware i.e. container-based virtualization. Docker provides ease of development and deployment for developers, as the application can be shipped with complete environment and deployed with minimal effort.

A container is loaded with the required configuration, environment and code to execute an application in isolated space. When the execution of application is complete the container can be dismantled freeing the resources just like any other application running in its user space. The environment, configuration and code are defined in Dockerfile, which can be used to create an application image.

The application image is the definition upon which container can be started. Now a days many software vendors and opensource community use Docker based images to ship their application. The images can be shared privately or using Dockerhub.

Dockerhub is a platform where docker based images of various applications can be downloaded from. These images are uploaded to Dockerhub by application developers or general public. In this work we have used SCONE in major parts of the implementation. SCONE provides a mechanism to run Docker based containers securely in trusted executed environments, which makes Docker an important pillar of this implementation.

3.3 Trusted Execution Environments

Trusted execution environment (TEE) [SAB15] is a reserved environment inside the processor which runs a piece of code securely, ensuring confidentiality and integrity of the execution. The TEEs ensure the secure execution over untrusted system, in presence of root admin as adversary. The protected region inside the processor is completely secure and no process other than trusted code can be executed in it. The authenticity of the TEE is established by the hardware vendor using specialized attestation mechanisms. After hardware is attested by the vendor upon boot up, trusted code can be loaded in it after its hash verification. In this case the hash of the code is calculated by TEE at bootup. Every interaction of trusted code outside the trusted execution environment, is end to end encrypted in order to ensure the confidentiality of execution. No application or root admin can peek into execution details.

Once the authenticity of the TEE and code is established it can be trusted that the application was executed privately in an overall untrusted surrounding. The TEEs are tamper-resistant and any attempt to update or hack the hardware renders them useless.

Multiple hardware vendors provide their implementations of TEEs. The most well-known implementations of TEEs are Intel's Software Guard Extensions (SGX) [Fuk19] [Sch16], AMD Platform Security Processor (PSP) [BWS19], ARM TrustZone [ARM] [Nga+], IBM Secure service container [Ros+18]. In our system Intel SGX based trusted execution environments were used, the details of which are discussed below.

3.4 Intel Software Guards Extension (SGX)

Intel's implementation of trusted execution environments also known as Intel SGX is a set of security related hardware instructions that are supported in modern Intel Skylake CPUs [HVH18]. These hardware instructions allow the user to create private execution regions called enclaves. The contents of enclaves are protected such that no other process or even root user can access them other than the enclave itself.

Enclaves have their own private memory known as Enclave Page Cache (EPC). EPC is a physically separate memory space and it is only accessible to enclave for its secure operations. The maximum size of EPC is 128 MB. If memory requirements of the workload exceed the EPC size, a memory paging mechanism is introduced in Intel SGX.

Memory Encryption Engine (MEE) in SGX encrypts the memory page that is supposed to be written to external memory accessible in user space i.e. DRAM. Similarly, when a memory page is loaded into enclave from DRAM it is decrypted for processing. Hence, providing such encryption and decryption mechanism in MEE, the memory of secure execution is protected in Intel SGX.

Just like memory encryption and decryption scheme managed by MEE, the applications running inside enclaves are expected to include such schemes for other unsecure operations. Such as, file system operations (that write and read from outside the enclave) must be encrypted while writing to disk and decrypted when loading from disk.

In order to run a program in secure enclave it must use the hardware instructions introduced in Intel SGX. To make the development of applications for running inside secure enclaves easier, Intel provides an SDK known as Intel SDK. The details of Intel SDK are discussed in next chapter.

3.5 Intel Software Development Kit (SDK)

Intel SDK [Cor] enables the developers to write C/C++ based code from scratch to run inside enclaves. The legacy C/C++ code cannot run inside enclaves. It needs to be updated to make it compatible to run inside enclaves, using Intel SDK. Besides, if a code is written in any other language, it cannot be adapted with minimal effort. Rather, it must be reimplemented in C/C++, which poses a great challenge for application developers. In order to run legacy applications inside Intel SGX and provide multiple language support, multiple frameworks like Graphene and SCONE are available.

3.6 Graphene

Graphene [TPV17] is a Linux compatible library OS that can support the execution of legacy applications inside Intel SGX. A typical library OS runs application facing libraries and required operating system modules in user space where a typical application is executed. As all the required modules for an application are loaded along with the application, it can run virtually in any environment. As long as the application specific modules are compatible with Host OS interface.

Graphene library OS is compatible with Intel SGX; hence it can load unmodified legacy applications along with their dependencies and run them in secure enclaves. This feature of Graphene provides it immense power over Intel SDK. Virtually every software and language stack can be supported using Graphene with Intel SGX, whereas in Intel SDK only C/C++ applications are supported, that too after code rewriting.

Graphene has also has a contender i.e. SCONE, which also supports to execute unmodified legacy applications. SCONE comes with a different underlying implementation, providing some architectural and performance benefits over Graphene which would be discussed in next section.

3.7 SCONE

3.7.1 Introduction

SCONE [Arn+16] is a secure containers technology that allows to run docker containers in trusted execution environments over Intel SGX. SCONE supports legacy applications, without any code change or instrumentation. In order to run legacy code, it needs to be recompiled by scone curated cross compilers, which generate scone curated runtime binaries. SCONE supports a wide range of SCONE curated application images for various applications like Memcached, Redis, Openvino etc. which can be fetched from official Dockerhub repository of SCONE. Multiple programming languages i.e. C, Java, Python, RUST, Go etc. are supported in SCONE.

SCONE provides a C standard library interface which leads to a small Trusted Computing Base (TCB) and small performance overhead as compared to Graphene, which implements a library OS.

3.7.2 SCONE Shields

SCONE implements some system level shields which ensure integrity and confidentiality of applications running inside secure containers. There are three types of shields in SCONE (1) Filesystem shield (2) Network Shield (3) Console Shield. These shields protect the data and traffic from operating system, hypervisor or any privileged process. In this thesis, SCONE file system shields have been extensively used to protect the code and configuration.

3.7.3 Remote Attestation

In order to trust the execution of an application running inside a secure enclave, the authenticity of code and hardware needs to be established, which is typically done by remote attestation. Attestation is handled differently in SCONE and traditional Intel SGX.

Intel Attestation Service (IAS) Intel provides a service that attests the hardware by verifying its group id that is allotted to the hardware by the vendor. This service is called Intel attestation service (IAS). Using IAS, we can get an attestation report from Intel which can be verified by the client. The attestation report contains information about the enclave which is used to determine the state of the system.

MRENCLAVE
MRENCLAVE is a 256-bit hash that is used to represent the identity of an enclave. Once an application code is loaded into an enclave, its MRENCLAVE is calculated from the contents of the code. If there is an update in the code or a malware is attached with the original code, the MRENCLAVE value would change. Hence, it can be determined if the enclave is running with the expected system state, by comparing the expected MRENCLAVE and generated MRENCLAVE. The expected MRENCLAVE should be known to the application verifier.

Attestation in SCONE
SCONE abstracts the traditional Intel SGX attestation mechanisms and provides transparent hardware and code attestation using SCONE's own Configuration and Attestation Service (CAS). Programs running inside SCONE based secure containers connect to CAS to get their secret configuration. CAS verifies the MRENCLAVE and quote of the requesting container before provisioning of the secret configuration, in a way CAS attests SCONE based secure containers.

3.7.4 Configuration and Attestation Service (CAS)

Configuration and Attestation service (CAS) is one of the most important components of SCONE. CAS itself runs inside a SCONE container and its attestation can simply be done via SCONE client. As CAS runs inside a trusted execution environment, its processing is completely secure. CAS has the capability to generate and import secrets, which it can securely provide to the SCONE containers booting up in context of CAS. CAS provisions these secrets to the SCONE containers in such a way that no unintended party can have access to these secrets.

Due to secure secrets provisioning in CAS, it enables the transparent remote attestation of the SCONE containers. Besides an attestation entity, CAS also acts as a certificate authority in a SCONE cluster and issues CAS generated TLS certificates to applications. SCONE containers can get private key of a certificate issued by CAS, only if their authenticity is verified by CAS. Using the public certificates from CAS, client can verify the authenticity of the SCONE containers by establishing a TLS connection. TLS connection would not succeed if the secrets were not properly provisioned by CAS. Implicit trust on a SCONE container is inherited by trust on CAS. This way, CAS enables transparent peer to peer attestation of the SCONE based secure containers without involvement of Intel's IAS.

CAS also ensures confidentiality and integrity of file system volumes in SCONE based containers. CAS manages the secret key and tag which can be used to decrypt and authenticate the filesystem. In case of an update in the filesystem by application, it transparently communicates with CAS to update the key and tags.

In our work CAS has been used extensively to implement following workflows:

1. Provisioning of secure secrets to SCONE containers.
2. To transparently manage file system encryption and authentication
3. It acts as a CA to provide TLS certificates, so other containers can attest and communicate with SCONE containers

3.8 Hyperledger Avalon

3.8.1 Introduction

Hyperledger Avalon [Avab] is one of the projects incubated at Hyperledger foundation, that enables creation of an ecosystem of trusted compute workers. These workers can securely execute custom workloads for various blockchain and non-blockchain based clients.

One of the main use cases of Avalon is to improve blockchains by solving two main problems i.e. privacy and scalability. Blockchains provide computational trust, but generally they are not privacy preserving in nature. Avalon helps to leverage the computational trust provided by blockchains while preserving privacy. It also helps to offload intensive computation from blockchains by executing complex workloads in trusted compute workers. Merged with Avalon, blockchains are used for transaction auditability, and Avalon is used for off-chain confidential execution.

Avalon provides its own independent workflow to submit requests to trusted workers, it's not bound to a particular blockchain. Besides blockchains, other non-blockchain based clients can also leverage the trusted execution of workloads provided by Avalon.

Avalon has two models:

1. **Proxy Model:** for blockchains connectivity using DLT bridge
2. **Direct Model:** for direct connectivity of clients using JSON RPC API

Avalon has a generic design, that supports multiple type of trusted compute workers i.e. Zero Knowledge Proofs (ZKP), Multi Party Compute (MPC) or Trusted Execution Environments (TEE). However, current implementation of Avalon only supports hardware based trusted execution environments (TEE) based on Intel SGX. Current supported TEE workers in Avalon are Intel SDK and Graphene. The development of Avalon ecosystem is not complete yet, and it is still in incubation. However, its pre-release version is available in their official GitHub repository. In this thesis, we have forked Hyperledger Avalon at "pre-release version 0.6" and made it compatible with SCONE based TEE workers.

3.8.2 System Overview

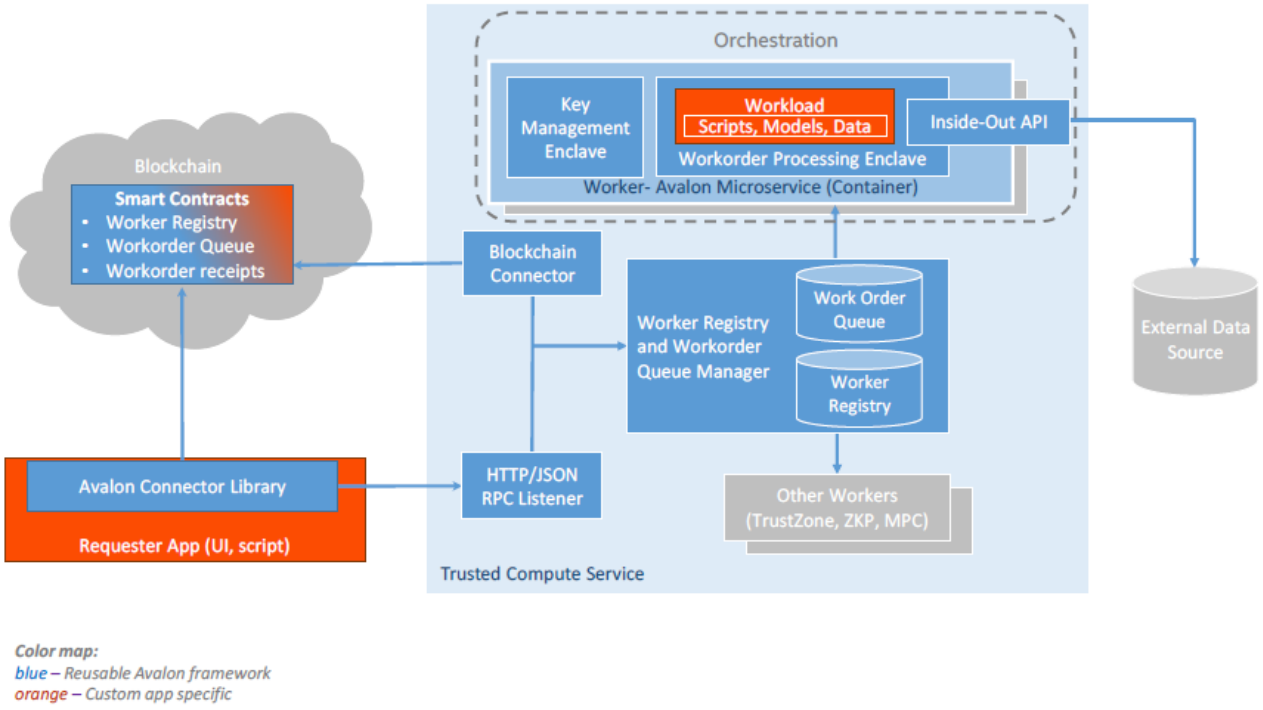

Figure 3.2 – Avalon system components [Avab]

3.2 shows high level system components of trusted compute workers eco-system provided by Avalon. The description of components is given below:

1. Workorders can be submitted to the trusted compute workers either by requester clients or by smart contracts.
2. Smart contracts use blockchain connector to submit workorders, whereas third party requester clients use Http listener service.
3. Worker registry contains the details of the available workers. It includes (1) attestation verification report (2) public RSA encryption key (3) public ECDSA SECP256K1 verification key for each worker.
4. The requests from blockchain connectors or listener service are forwarded to workorder queue managers, after fetching the worker details from the worker registry.
5. Worker queue managers forward the request to the workers for processing and return the response from workers back to the clients.
6. Key management enclaves are responsible for generating encryption and verification keys for workers and securely transmitting these keys to the workers running in secure enclaves.

7. Workers also known as Workorder processing enclaves contain the workloads to be processed. Upon receipt of the request they execute the relevant workload and return the response. The workloads can either be precompiled or can be sent as part of request (in case of interpreted language i.e. python or solidity).

3.8.3 High Level Execution Flow

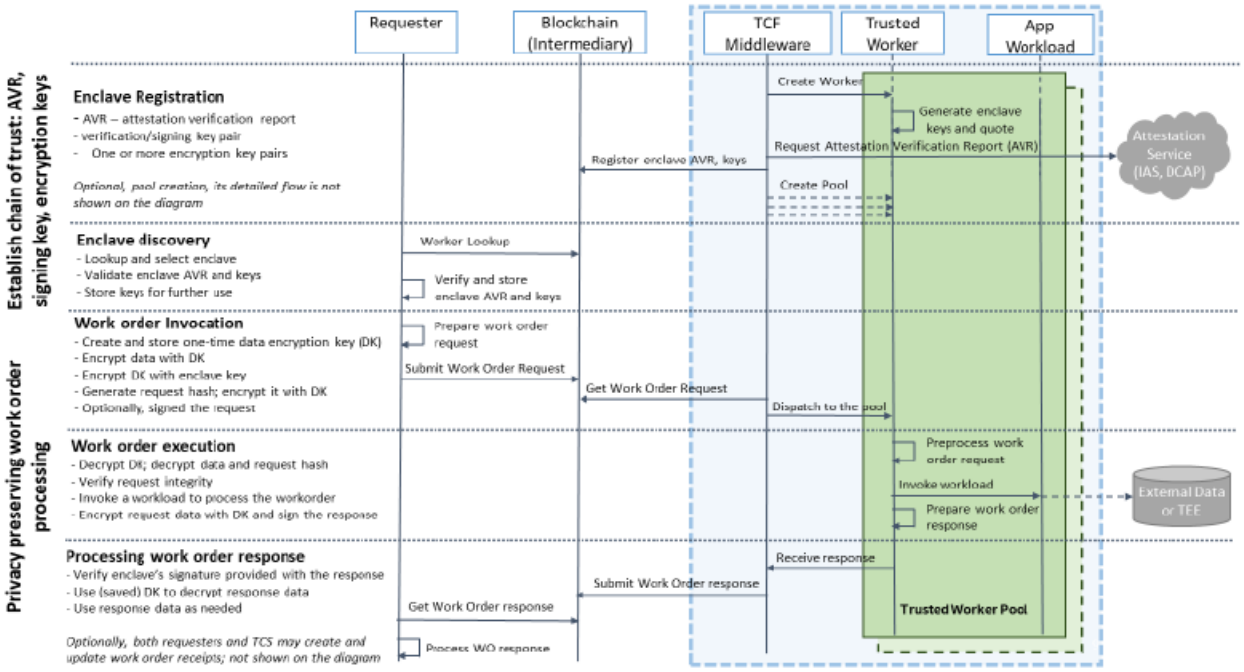

Figure 3.3 – Avalon flow diagram [Avab]

3.3 shows high level flow diagram of Avalon ecosystem. The generic workflow is explained below:

Enclave Registration

A new worker bootstraps in first step. The worker creates two key pairs i.e. Encryption key pair and Verification key pair. The worker attests itself at Intel attestation service (IAS or DCAP) and a corresponding attestation verification report (AVR) is generated. The AVR is signed by Intel's IAS. The verification report contains enclave specific report data, i.e. hash of the verification public key and quote info. This signed AVR, along with the workers basic information i.e. name, id, public verification key, public encryption key, verification encryption hash etc. is published in the registry service. The registry service in this case could be blockchain or any other intermediary.

Only the public keys and their attested proof (AVR) leaves the enclave. The private part of the keys remains in the sealed storage.

Enclave Discovery

The requester searches for the worker in directory service. If found, the worker information is fetched. The requester verifies its attestation information. If attestation information is verified the public encryption and verification keys are extracted. The authenticity of public keys is checked against the hash of verification public key extracted from AVR. If authenticity of the keys is verified, they are saved for further use.

Work order Invocation

The requester creates a workorder processing request and generates a symmetric key. The symmetric key, request and its hash are encrypted with the public encryption key (extracted in the previous step). The resultant encrypted workorder processing request is sent for execution.

Work order Execution

The work order processing requests is retrieved by worker. The worker decrypts it, using private part of the encryption key and extracts the request and the symmetric key. The request is then processed in worker according to the requested workload. The response is encrypted with the symmetric key and signed by the private part of the verification key.

Processing work order response

The encrypted response is retrieved by the requester. The signature of enclave is verified using public verification key. The response is decrypted using the symmetric key, which was earlier generated by the requester while sending the request.

3.8.4 Architecture

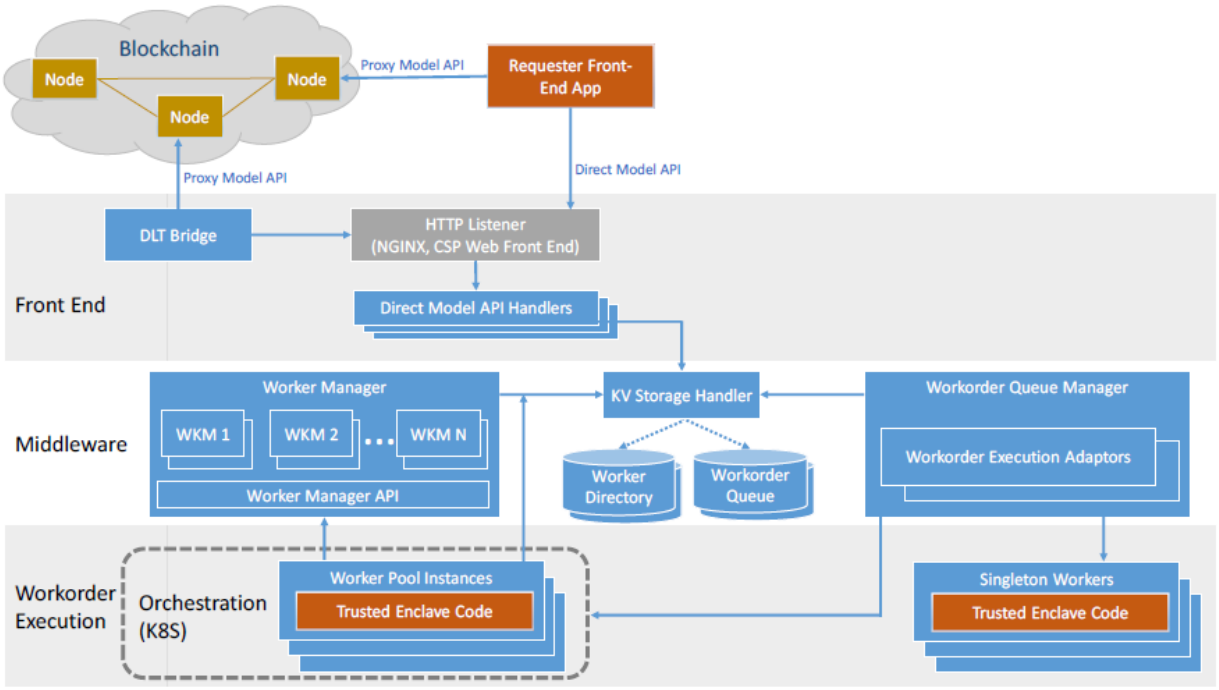

Figure 3.4 – Avalon architecture diagram [Avab]

Figure 3.4 shows architecture diagram of Avalon ecosystem. The architecture is explained below.

As shown in the Figure 3.4, Avalon is divided into three tiers:

1. **Front End:** This tier bridges the Avalon infrastructure with the clients
2. **Middleware:** This tier consists of KV storage handler, worker managers, worker queue managers
3. **Workorder Execution:** This tier contains various trusted worker pools

Front End

Front-end tier consists of two important services in Avalon infrastructure.

1. **Distributed Ledger Technology (DLT) Bridge**
   This service is used by various blockchains to connect to Avalon infrastructure in a so-called proxy model. Avalon provides connectors for different blockchains, which are leveraged by blockchains to send requests using DLT bridge.
2. **Listener Service**
   This service is used by non-blockchain based third party clients to send work order requests in a so-called direct model.

Multiple instances of these services can be bootstrapped to provide scalability and NGNIX load balancers can be added in production like environment.

Middleware

This tier manages the communication between the front end and work order execution layer. It consists of three main services.

1. **KV Storage Handler**
   KV storage handler is a service which manages worker directory and workorder queue. (1) Worker directory contains the information about the available workers which are waiting to receive and process the requests (2) Workorder queue contains the pending workorder requests and completed workorder responses.
2. **Worker Manager**
   Worker manager pre-processes the requests and manages the worker key managers (WKM). Worker key managers create and manage the private keys for workers and workers get these keys upon bootup.
3. **Workorder Queue Manager**
   Workorder queue managers, are responsible for pre-processing and submitting a work order request to the trusted workers. Unlike worker managers they do not support worker key managers, instead the workers generate and manage their keys themselves. Workorder queue managers are particularly used with singleton workers.

Workorder Execution

This tier consists of trusted compute workers in Avalon. There are two types of workers:

1. **Singleton workers:** Only one worker on a physical system, which generates its own keys
2. **Worker pools:** Multiple worker instances deployed across a distributed cluster, have worker key managers to generate and provision secret keys

There can be varying types of clusters i.e. Docker swarm clusters, Kubernetes clusters etc. In this tier multiple type of workers can be added, such as Intel SDK workers, Graphene workers, SCONE workers, ZKP workers, MPC workers etc.

3.8.5 Avalon Modes

In Avalon the workorders can be submitted in either synchronized mode or asynchronized mode.

1. **Synchronized Mode**
   The workorder queue managers listen for requests at their web sockets. In case a request is received they forward the request to the corresponding worker and return the response when they receive it from the workers.

2. **Asynchronized Mode**
   The request is added on the workorder queue and the respective managers pick the requests in an asynchronized manner i.e. by polling the workorder queue after every 'n' seconds. If the request is available in the queue, it is forwarded to the worker for execution. If a response is received, it is also added on the workorder queue for client to pick it.

4 Design

In this chapter we describe the design of the solution presented in this thesis. The opensource code of Hyperledger Avalon is forked to support SCONE based trusted compute workers in this work. We would discuss the system overview and design goals of the system. Furthermore, we discuss the threat model, security aspect and architecture of the system. Avalon's design and architecture explained in previous chapter is a pre-requisite to this chapter. In order to completely understand the upcoming chapters, Avalon and SCONE topics from previous chapter should be consulted.

4.1 System Overview

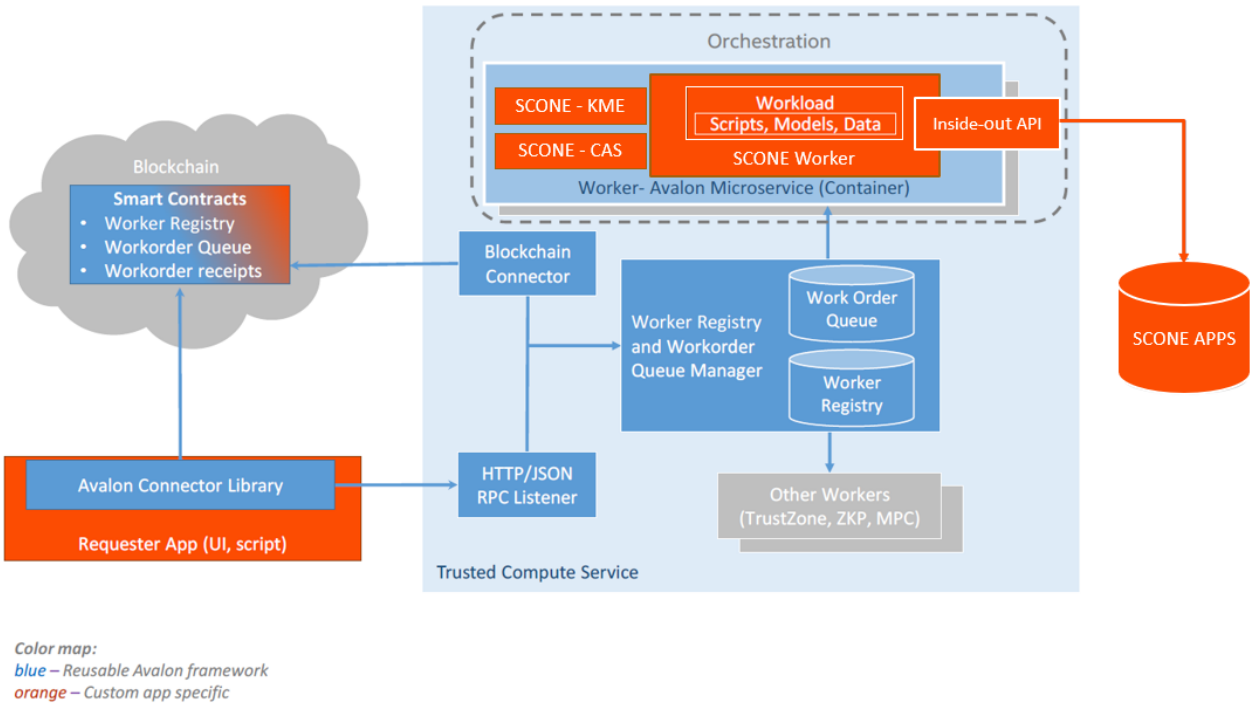

Figure 4.1 – Avalon with SCONE system components

Figure 4.1 shows a high-level overview of the forked Avalon ecosystem. The components of Avalon ecosystem discussed in previous chapter remain as-is. The support for SCONE based workers and its configuration and attestation service (CAS) is added in the orchestration layer. The SCONE based components, which have been added in existing eco-system of Avalon are dis-

cussed as following:

SCONE Configuration and Attestation Service (CAS)
SCONE's configuration and attestation service (CAS) is an integral part of the SCONE platform. In our work, CAS enables following workflows:

1. It provides transparent attestation and secret provisioning to SCONE based trusted containers. The requester clients use CAS for transparent attestation of SCONE based workers.
2. It acts as a certificate authority (CA) and generates TLS certificates for SCONE containers, which are used for their communication with SCONE based external apps.
3. It is used for managing filesystem protection keys and tags in SCONE based workers.

SCONE Key Management Enclave (KME)
Key management enclaves (KME) is a SCONE based trusted container which generates the encryption and verification keypairs for SCONE workers. These keypairs are used by clients to encrypt the requests sent to workers and verify the authenticity of the response. SCONE KME publishes these keypairs on CAS which are provisioned to the workers on bootup.

SCONE KME also exposes an API, which enables the clients to verify the SCONE workers and also provides their CAS session details.

SCONE Worker
SCONE worker executes the requested workload securely in Intel SGX based trusted execution environments in Avalon ecosystem. The workloads can either be precompiled or can be sent as part of request (in case of interpreted language i.e. python or solidity). SCONE workers can also be connected to SCONE based external applications. The peer-to-peer secure communication and attestation is enabled by CAS.

4.2 Design Goals

The goal of this work is to create an ecosystem of trusted workers which can run legacy applications without any code changes or instrumentation. The resulting ecosystem should support various clients, especially blockchains which can be made scalable and privacy preserving in conjunction with this work. To enable such a system, we have made certain design level decisions which derive our architecture. The design goals of the system are explained as following:

Extensibility

The workers are generic, with a minimum functionality of pre-processing a request. They should be extensible so that new custom workloads could be added with minimal effort.

SCONE provides a wide range of scone curated applications on the Dockerhub. Apart of adding custom workloads in worker, there must be a way to integrate SCONE workers with external

SCONE apps. So that the true potential of SCONE can be utilized with minimal effort.

Attested Workers and KME
The SCONE based secure containers i.e. workers and key management enclaves should be transparently attested using CAS.

Secure Provisioning of Secret Keys
The secret keypairs are generated in SCONE KME, these keypairs are used by SCONE workers to securely communicate with the clients. These keypairs must be confidentially transferred to SCONE workers.

Secure Management of Secret Keys
The secret keys should be independent of the workers, managed by a trusted entity or code. In case of worker restart, they could be fetched again and if worker goes offline they could be provisioned to a newly spawned worker with same MRENCLAVE.

Attested Worker Keys
The worker's public keys are used to establish secure communication with it. If an adversary changes the keys and replace them with its own keys, it can lead to man in the middle attack. There must be a way to attest the authenticity of these keys, which are fetched from worker registry.

Scalability
The system should be scalable, it should support worker pools and orchestration using either Kubernetes or Docker Swarm.

Backward Compatibility
The system should ensure backward compatibility. It should not affect the current workflow of Avalon ecosystem and the SCONE based workers should work in conjunction with other workers supported by Avalon.

4.3 Threat Model

Up till now we have discussed Avalon's detailed architecture and high-level overview of our solution i.e. Avalon ecosystem with SCONE workers. The attack surface of the proposed system with SCONE workers is same as the original Avalon design. Avalon's original implementation depends upon Intel's attestation for various security related workflows. However, in our system we have used SCONE's transparent attestation with CAS; it leads to some SCONE specific design goals, as discussed in section 4.2. Due to the updated design, the way we handle attestation and keys has changed.

In our threat model, we consider an omnipotent adversary who can access all communications, disk spaces, and have root admin access of the complete ecosystem. Not only this adversary can read the information, but can also write and update the information in workorder queue, worker

registry etc. It can also actively participate in communication.

As SCONE containers are being executed inside trusted execution environments, they are considered a safe zone. The adversary cannot peek into enclave executions or read the data, file system or network communication encrypted by the enclave.

Attack Surface

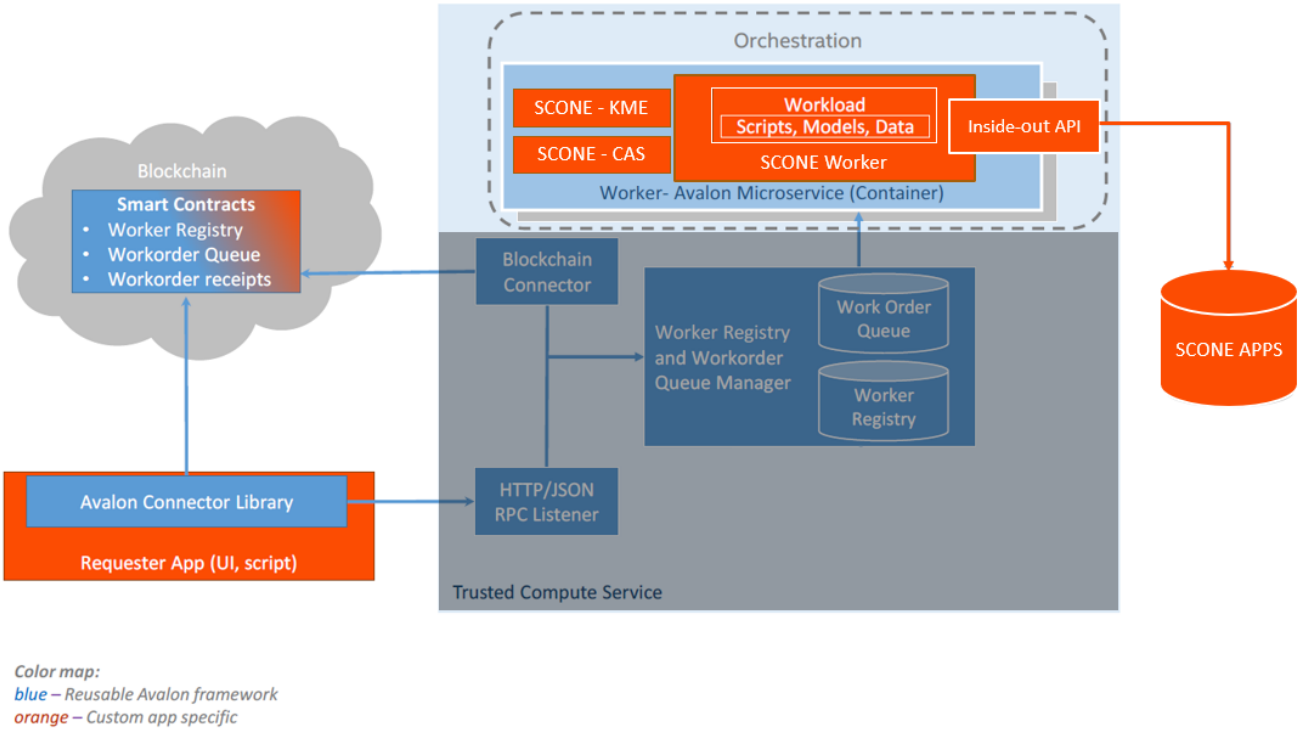

Figure 4.2 – Avalon with SCONE attack surface

Figure 4.2 describes the attack surface in presence of the omnipotent adversary as described above. The dark highlighted components of Avalon ecosystem in Figure 4.2, is considered in the control of adversary.

The SCONE containers and their communication (shown in orange colour) are not in control of adversary as they are being executed in enclaves. Their communication and filesystem are encrypted.

The security of client is out of scope for Avalon and it is assumed that the client implements its own security. The communication of client with Avalon (shown in blue arrows) is also unsecure and is in control of adversary.

Security Goals
In presence of an omnipotent adversary as described above, our goal is to ensure the secure execution of workloads, protection of requests and response messages across the ecosystem. The security goals of our system are:

1. **Confidentiality** The confidentiality in the execution of workloads and handling of input requests, workorder response, secret keys must be ensured.
2. **Integrity** The integrity of worker registry, workorder queue and network communication in Avalon ecosystem must be preserved.

4.4 System Architecture

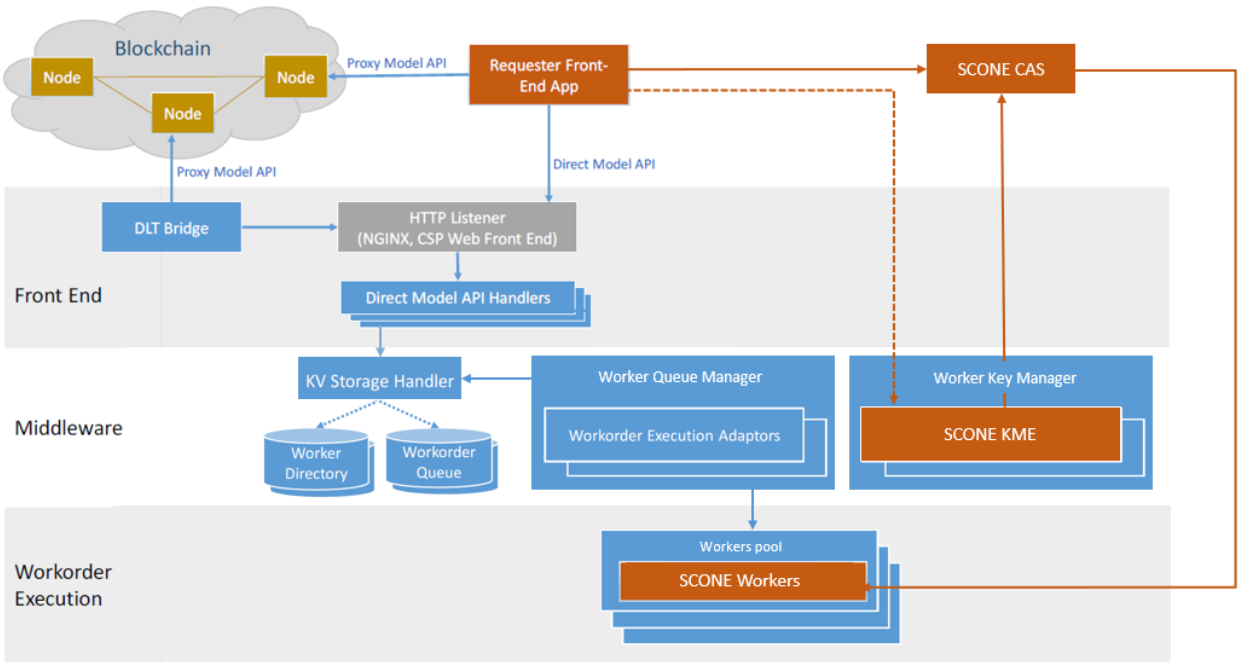

Figure 4.3 – Avalon with SCONE architecture

Figure 4.3 shows the architecture of the forked Avalon ecosystem, that supports integration with SCONE. As discussed in the overview section 4.1 the main components of Avalon ecosystem remain the same, only SCONE based workers and key manager have been added with CAS support.

CAS is used off the shelf, whereas SCONE workers and SCONE KME are our own implementation. SCONE CAS, SCONE KME and SCONE workers, all run inside trusted execution environments.

SCONE KME Workflow
SCONE KME is responsible for generation of encryption and verification keypairs for SCONE workers. The keypairs used in Avalon are (1) RSA for encryption (2) ECDSA SECP256K1 for signing and verification. The workflow of SCONE KME is explained as following:

1. Trusted setup creates a session for KME at CAS. This session includes the expected MRENCLAVE of KME.
2. KME gets attested at CAS and gets Transport Layer Security (TLS) certificates from CAS at bootup in context of CAS session.
3. KME securely generates the keypairs for each worker and publishes them at CAS as session secrets, so that they could be provisioned to their respective workers after their attestation.
4. KME publishes a REST API over SSL that provides CAS session ids for each worker, so clients can optionally get the session ids for the workers.

SCONE Worker Workflow
SCONE workers are responsible for processing the requested workloads in trusted execution environments. Using Avalon, the workorder processing request is forwarded to the workers via workorder queue managers. The workflow of SCONE workers is explained as following:

1. SCONE workers bootup in context of CAS sessions and after their attestation CAS provides them their respective secrets.
2. After receiving the secrets securely from CAS, the SCONE workers initialize themselves and register at worker registry. Only basic information about the workers i.e. name, id, organization id etc. is published on worker registry in contrast to traditional Avalon. In traditional Avalon design, the public keys and Intel's signed attestation report (AVR) is also published in the registry.
3. After registration, the workers wait for the workorder processing requests.

Transparent Attestation Using CAS
Requester clients use CAS for establishing the authenticity of SCONE KME and SCONE workers. There are two workflows available for the requester clients (1) The session of SCONE workers is known to requester (2) The session of SCONE KME is known to requester. Both these workflows are discussed below:

1. The session of SCONE workers is known to requester
In this workflow it is assumed that the session ids of the SCONE workers are available publicly and requesters know the exact CAS session for each worker. In this case SCONE workers can be attested by the requesters as following:

1. Requesters attest CAS by using SCONE client

2. Requesters call the REST API of CAS over SSL to get public keys of SCONE workers for a particular session

3. Requesters use these public keys to encrypt the request and verify the authenticity of the response. Only the correct worker can decrypt the request and provide a response. Hence, the worker's attestation is transparently enabled by CAS

2. The session of SCONE KME is known to requester

In this workflow it is assumed that multiple workers of a particular organization or owner are running in a distributed cluster. Each may have its own MRENCLAVE and workloads. The key manager enclabe (KME) not only generates the keypairs for workers, it also represents the cluster. Instead of publicly publishing session ids for each worker, the owner can just publish the session id of KME. If the authenticity of KME is established via CAS, it can be connected with, to get session ids for any of its workers through its API.

It solves the hassle of publishing the session ids of each worker and managing the change in sessions ids for cluster owners. This design improves the management overhead for traditional Avalon worker pools. The worker attestation workflow in this mode is explained as following:

1. Requesters attest CAS by using SCONE client

2. Requesters get CA signed cert of KME from CAS

3. Requesters call the REST API of KME over SSL to get the session id for a particular worker

4. Requesters call the REST API of CAS over SSL to get public keys of SCONE workers for a particular session

5. Requesters use these public keys to encrypt the request and verify the authenticity of the response. Only the correct worker can decrypt the request and provide a response. Hence, the worker's authenticity and attestation are transparently enabled by KME and CAS

Workload Processing

In this design we have used SCONE to enable attestation and key management, the rest of the design of Avalon ecosystem remains the same. Hence, the request and response workflow are the same as explained in Avalon's architecture in section 3.8.4. A brief overview of request and response workflow is given below:

1. Requester generates a symmetric key.

2. Requester encrypts the request and symmetric key using worker's public encryption key and sends it using Avalon ecosystem.

3. Worker gets the encrypted request from workorder queue manager.

4. Worker decrypts the request and symmetric key using the private part of its encryption key.

5. Worker executes the requested workload securely.

6. Worker signs the response by its private signing key and encrypts it using the symmetric key, exchanged in step 2.

7. The user decrypts the response using the symmetric key exchanged in step 2 and authenticates the response using workers public verification key.

4.5 Security Aspect

In our design, ensuring integrity of the worker registry, workorder queue and network communication is of primary concern as the adversary is in total control of the attack surface as discussed in section 4.3.

In traditional Avalon design, the worker registry contains the information about the registered workers, quote, report data and public keys. The integrity of worker specific information is ensured by adding attestation verification report (AVR) along with the worker. It contains hash of the worker specific details and it is signed by the Intel's IAS. The client checks the authenticity of report by checking intel's signatures, and then validates the worker information by comparing them with the contents in report data. This way, the integrity of worker registry is established in classic Avalon design. Avalon depends upon Intel's IAS verification for ensuring integrity in the ecosystem.

In our implementation we have not used Intel's attestation service and have replaced it with SCONE's CAS. Hence, the integrity of the information on worker registry is established using CAS and KME. Minimalistic information is added on worker registry i.e. worker name, id, organization id and some other basic information. The session id for a worker is either publicly published, or it can be fetched from KME. Worker registry is not used for sharing any sensitive information.

Once the requester has the session id of a worker, it can use this session id to get the public encryption and verification keys from CAS. The request and response published on the workorder queue are also thoroughly encrypted.

While the attacker has the absolute control over the worker registry, workorder queue and communication. The security goals of the system i.e. confidentiality and integrity are fulfilled as discussed above.

5 Implementation

In this chapter, implementation details and the toolchain of the Avalon ecosystem are discussed. We talk about the system configuration, worker pools and implementation details of the SCONE curated images of KME and Worker. Furthermore, the steps required to bootstrap the ecosystem with a cluster of 'N' SCONE workers, adding custom workloads, integration with SCONE based external apps and some examples developed as part of this thesis are also discussed.

5.1 System Configuration

This system has been developed and tested on a virtual machine over a server with SGX enabled Intel CPU Xeon E3-1280v6 @ 3.90 GHz, 62 GB RAM and 8 processors. The operating system was ubuntu 18.04.3 LTS and docker version was 19.03.6.

Hyperledger Avalon has been forked at latest stable pre-release version 0.6. SCONE version was 5.0 and SCONE CAS and LAS version 4.2.1 was used.

For the creation of SCONE curated images of SCONE Worker and SCONE KME, various libraries were cross compiled using toolchain provided by SCONE. The libraries and their respective versions are listed as following:

Library	**Version**
Libzmq	4.2.2
Libffi	3.3
Libressl	3.0.0
Python	3.6.12
Alpine	3.7

Table 5.1 – Library versions

5.2 Avalon with SCONE Cluster

The system consists of a cluster of Avalon and SCONE based services. The services in the cluster and their purpose are explained as following:

1. **SCONE CAS**
   SCONE CAS is a configuration and attestation service provided by SCONE. It enables transparent attestation of SCONE Workers and KME, secret keypairs provisioning to SCONE Workers, acts as a CA for SCONE Workers and KME. SCONE CAS has been used off the shelf i.e. without any customization.

2. **SCONE LAS**
   SCONE LAS is a service that provides local attestation and quotes to SCONE based containers. Like SCONE CAS, it is also used off the shelf.

3. **Avalon Shell**
   It is a container provided by Avalon that can be used to send workorder processing requests to Avalon ecosystem. This container has pre-configured environment for the Avalon clients. It is part of Avalon's implementation and used as-is.

4. **Avalon LMDB**
   It is key value storage handler that acts as worker directory and workorder queue. It is used by workorder queue managers to pick and put request and responses, and by workers to register themselves. It is also part of Avalon's implementation and used as-is.

5. **Avalon Listener**
   Avalon listener is a front-end service, that listens for the request from clients, writes them on the workorder queue and similarly provides the response. It is also part of Avalon's implementation and used as-is.

6. **Avalon Enclave Manager**
   This service implements Avalon's workorder queue manager, it enables the communication of SCONE workers with the Avalon ecosystem. It has the following tasks:

   a) Registers SCONE workers on the worker registry
   b) Forwards the requests from workorder queue to SCONE workers
   c) Forwards the response from workers back to workorder queue.

   The Enclave manager provided in Avalon currently supports only one worker at a time. We have forked the Avalon enclave manager provided in Avalon and updated it (1) To work with SCONE based workers (2) To support pool of 'N' SCONE workers.

7. **SCONE KME**
   SCONE KME is the key manager which generates and manages secret keys used by SCONE workers. The details of SCONE KME is discussed in detail in section 4.4. It is implemented in python. There can be multiple KME containers in a cluster, each managing keys for its pool of workers.

8. **SCONE Worker**
   SCONE worker is used to execute the confidential workloads in TEEs. The detail of SCONE worker is discussed in detail in section 4.4. It is also implemented in Python. There can be many workers in the pool, executing multiple workloads in parallel. In our demo we have a cluster of five SCONE workers, but they can be increased or decreased as per requirement.

5.3 SCONE Curated Images

In this work we have created a SCONE curated docker image using SCONE toolchain, that installs all the dependencies and sets up environment required to run our applications. This image is called 'avalon-scone-dev'.

avalon-scone-dev is a baseline image, that can be further enhanced. We further add the code of KME and worker in it to generate their respective SCONE curated images i.e. 'avalon-scone-kme-dev' and 'avalon-scone-worker-dev'. These images are then used to bootup the SCONE KME and SCONE Worker containers.

1. **avalon-scone-dev**
   Docker's multi-staged build is used to create this image. The Dockerfile had two stages:

   Stage 1: FROM sconecuratedimages/muslgcc:alpine-scone5

   In stage 1 we cross compile Libzmq, Libffi, Libressl, Python and install all the require python packages.

   Stage 2: FROM sconecuratedimages/crosscompilers:runtime-alpine3.7-scone4

   In this stage we copy the required libs to create scone runtime image. The resultant image is lightweight, as it contains minimum required cross compiled libraries. It can be used to generate further images.

2. **avalon-scone-kme-dev**
   The code of key manager is added in the baseline 'avalon-scone-dev' image. Encrypted code regions and authenticated dynamic libraries regions are created to generate a secure SCONE curated image for SCONE KME. The Dockerfile has only one stage:

   Stage 1: FROM avalon-scone-dev

3. **avalon-scone-worker-dev**
   The code of worker is added in the baseline 'avalon-scone-dev' image. Encrypted code regions and authenticated dynamic libraries regions are created to generate a secure SCONE curated image for SCONE Worker. The Dockerfile has only one stage:

   Stage 1: FROM avalon-scone-dev

   The SCONE curated images generated in above section consist of SCONE runtime, cross compiled libraries and protected file system. They can be used to bootstrap the containers

by using simple docker commands. New SCONE workers can be added by editing the docker-compose file and updating the config files provided in the demo. Distributed clusters using docker swarm and Kubernetes can also be generated easily.

5.4 Custom Workloads

The system is designed so that it is extensible, i.e. (1) new workloads can be added (2) the system could also be integrated with SCONE based external applications. These workflows are explained as following:

1. **Adding New Workloads**

   The SCONE worker is implemented in python. A simple interface is provided, so that new python-based workloads can be added easily. If workload is in some other language like C/C++, then its dynamic library cross compiled by SCONE should be used with python-based workload. We have developed some workloads in SCONE workers as examples:

   a) **Echo**
   It is a hello-world workload. It appends "hello" with the input string and returns the appended string.

   b) **Fibonacci**
   In mathematics, Fibonacci is a series of numbers, such that N^{th} Fibonacci number is sum of two preceding Fibonacci numbers. In computer science Fibonacci is an interesting workload, as it has iterative and recursive implementations with complexities O(N) and O(N^2) respectively. We have provided Fibonacci implementations as examples in our work. These implementations are also used for benchmarking.

   c) **Secure Discounted Transaction**
   It is a demo of real-world use case of coins transfer from one wallet to another wallet. In blockchains, the execution of smart contracts is not confidential, and the code written in smart contract is visible to participants in network. It is not possible to provide variable discounts to different customers without the other customers knowing about it.

   Using trusted execution, different discount voucher can be provided to each customer without anyone knowing about it, other than the two parties involved. This workload can be integrated with smart contracts using blockchain connectors to enable trusted confidential transactions.

2. **Integration with External SCONE curated apps**
   SCONE provides a wide range of SCONE curated images of various applications on Dockerhub. We want to reuse the existing applications provided by SCONE. Hence, we have provided a few examples of a workflow of integrating the ecosystem with external SCONE based apps. SCONE CAS is used for transparent peer to peer attestation and enabling communication between SCONE Worker and external apps. Hence, we provide following examples:

   a) **OpenVino**
      Openvino is a machine learning based computer vision tool. Based on trained models, it can detect objects in an image or a video stream.

      In our example, we have added vehicle detection model in SCONE curated Openvino image, fetched from Dockerhub. If images and videos of security camera are provided to the workload it can detect the vehicles and their licenses. SCONE worker is integrated with this external app via CAS as explained above.

      It is a valid real-world usecase with the aspect that such computation intensive applications are difficult to develop and costly to execute in blockchain based smart contracts. Hence, we use Avalon ecosystem with SCONE to serve as an attested oracle for blockchain.

   b) **Hospital Patient Management System**
      This example demonstrates a hospital app running inside SCONE based secure containers. The patient's data in a hospital is critical with respect to privacy. Hence, the hospital app consists of a database and API running inside SCONE secure containers. It exposes patient's health score anonymously.

      In the example scenario, the hospital now wants to be part of a wider Avalon ecosystem or integrate its backend with blockchain. We have integrated it with Avalon ecosystem using SCONE workers. A client of this app is written as a workload and the worker is integrated with hospital app using CAS. The hospital app now acts as an attested oracle for blockchain.

6 Evaluation

As this is a complex system which creates an eco-system of trusted compute workers, the benchmarking has been done from three different aspects. The behaviour of different modes of SCONE, it's comparison with other trusted computing frameworks and scalability impacts in a pool of workers have been analysed for linear and exponential workloads.

6.1 System Specification

This system has been evaluated on an SGX enabled server which has Xeon E3-1280v6 @ 3.90 GHz CPU, 62 GB RAM and total of 8 processors. The operating system was ubuntu 18.04.3 LTS and docker version 19.03.6 was used to run the containers.

6.2 Evaluation Modes

The following modes of execution have been benchmarked in different experiments.

1. **Native Execution**
   This mode specifies that the execution of the workload is in native runtime without any trusted enclave, as if it was running in a machine without trusted execution environment.

2. **Simulation Mode**
   This mode specifies that the work order processor is behaving as if it is being run inside a trusted execution environment, whereas in fact it is running on a simple processor. Using this mode realistic applications can be developed without a physical SGX enabled hardware. We can test features and execution runtime with closest estimate.

3. **Hardware Mode**
   Hardware mode means the attested code is being executed in a trusted executed environment on an SGX enabled hardware.

4. **Hardware Mode with File System Protection using CAS**
   While the hardware mode protects the work order execution in memory, the code and file system of a typical SGX application remains unencrypted on disk. Depending upon the security goals of the application, this could be a security concern. Confidentiality can be

achieved by encryption of file system so that the code and other related input data from disk can only be accessed from inside the enclave. Similarly, integrity of application data and code could also be ensured by protecting the relevant hashes inside the enclaves. When SCONE workers boot up securely in context of CAS their relevant secrets are transmitted to them by CAS. This enables the worker to access avalon specific keypairs as well as keys to access filesystem. Hardware mode with end-to-end file system protection in context of CAS is an important workflow of our implementation of SCONE workers running inside Avalon ecosystem.

6.3 Evaluation Workloads

Following workloads have been used for benchmarking the system in various comparisons.

1. **Fibonacci Linear Workload**
   This workload calculates the sum of N^{th} Fibonacci number using iterative method. The latency of the workload increases linearly as the size of N grows. For the purpose of evaluation the traditional Fibonacci iterative algorithm is tweaked so that in each iteration an array of size N is initialized in heap and file system read and write are also added so that we have a workload that occupies N^2 memory slots and does N file system operations. It's space complexity is O(1) and time complexity is O(N).

2. **Fibonacci Exponential Workload**
   This workload calculates the sum of N^{th} Fibonacci number using recursive method. The latency of the workload increases exponentially as the size of N grows. For the purpose of evaluation, the traditional Fibonacci recursive algorithm is also tweaked on the pattern stated above. In each recursive call an array of size N is initialized in heap and file system read and write are also added so that we have a workload that occupies N^2 memory slots and does N^2 file system operations. Overall, it's space and time complexity is $O(N^2)$ each.

6.4 Evaluation Strategy

In this thesis, SCONE based trusted compute workers have been added in existing eco-system of Hyperledger Avalon. The main goal of the evaluation is to benchmark the performance of the workflows related to work order execution in SCONE based workers in Avalon.

Avalon eco-system consists of listener service, registry service, enclave managers and workers. The request processing can be done in either synchronized mode or asynchronized mode. In synchronized mode, a request can be sent directly to workorder queue manager via zmq socket, by fetching their respective worker details from registry service. Whereas in asynchronized mode the encrypted request is added in the worker registry and listener service subscribes for the response. The workorder queue manager keeps polling the worker registry after every few seconds to check if there is a request that needs to be processed.

In this evaluation we have calculated the average time taken for the Avalon infrastructure to create and send the request to the worker and to post-process the response in synchronized mode. There were no network delays and the average time is calculated by running all nodes in the same VM. The time taken by the workers for request processing is calculated separately and the estimated average time of Avalon infrastructure is added in that time to estimate end to end latency of the system in synchronized mode.

An application can be deployed inside Avalon workers if it is supported by the respective worker type. In our implementation of Avalon worker, python stack compatible with SCONE is provided so python based complex applications can be written. Besides this, the SCONE workers can also be integrated with external applications using secure communication enabled by CAS. As a part of this thesis, due to time constraint a simple workload which calculates Fibonacci of any number 'N' is chosen. It can easily be extended to low memory and high memory workloads and can easily be developed in other trusted execution environments like Graphene and Intel SDK for comparison. However, as future work evaluation of complex real-world application can also be done.

The three different aspects in which the performance of SCONE based workers have been measured are (1) Performance comparison of SCONE with other pre-existing workers in Avalon i.e. Intel SDK and Graphene (2) Performance benchmarking of various SCONE modes (3) Performance benchmarking of scalability in SCONE based worker pools. These benchmarking aspects and their results are discussed as following.

1. **Performance comparison of SCONE compared with other TEEs**

   In these experiments we aim to measure the performance of SCONE based workers in Hyperledger Avalon and analyse them in contrast to other trusted workers supported by Hyperledger Avalon, namely Intel SDK and Graphene.

   Intel SDK [Cor] is official development kit provided by Intel. It can be used to write application workloads runnable on Intel SGX hardware in trusted execution environment. Intel SDK based workloads are written in C/C++ and various starter applications are provided in Hyperledger Avalon official git [Avaa]. For the sake of benchmarking, C++ variants of the evaluation workloads discussed above i.e. Fibonacci recursive were used.

   Graphene [TPV17] is a libOS that allows isolated execution of applications inside Intel SGX secure enclaves. Like SCONE, Graphene also enables secure execution of legacy applications inside Intel SGX. We run python stack as a legacy application inside SCONE and Graphene both, which allows us to run python implementation of evaluation workloads.

   In subsequent experiments, average of the response time is calculated after measuring the batch of 1000 requests. SCONE, Intel SDK and Graphene are benchmarked in their respective hardware modes without file system encryption.

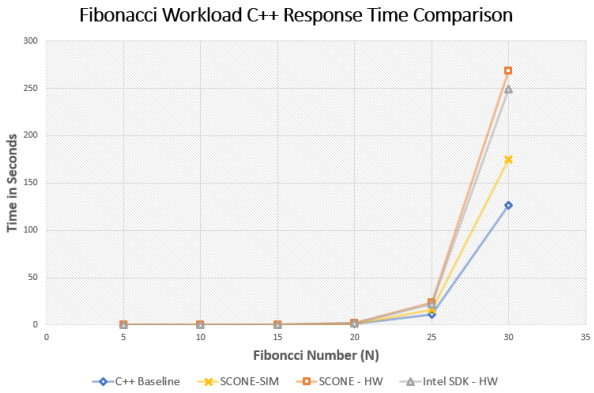

Figure 6.1 – Fibonnaci workload C++ response time comparison

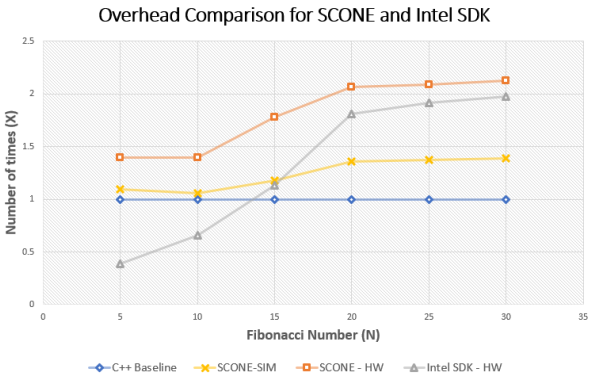

Figure 6.2 – Overhead comparison for SCONE and Intel SDK

- **Experiment: SCONE comparison with Intel SDK**

In this experiment the performance of SCONE worker is compared with Intel SDK for the C++ implementation of Fibonacci Exponential workload and their respective latencies and overheads are presented. In order to run C++ code in SCONE, the workload was built by musl gcc and was imported in python as a library. The benchmarking was done for C++ baseline, SCONE simulation mode, SCONE hardware mode and Intel SDK hardware mode. For this experiment, SCONE configuration with 2 enclave threads and 6 system threads with increased system and enclave thread spins (SSPINS and ESPINS) was used, as it was a sequential workload with exponentially increasing system calls.

Figure 6.1 shows response time comparison for various execution modes for Fibonacci of increasing 'N'. As expected, trend of the exponential workload is observed in each mode. For highest number 'N' the best performance was achieved by native C++ which also acts as baseline and the performance of SCONE simulation mode is slightly less than baseline. The latency of Intel SDK and SCONE hardware mode is close to each other, with Intel SDK having slightly better performance than SCONE.

In Figure 6.2, the overhead comparison of SCONE simulation, SCONE hardware, Intel SDK hardware modes with C++ baseline is presented. It can be observed that for very small workloads, the performance of Intel SDK with hardware is even better than native C++ code. This is because of a smaller number of system calls and low memory overhead in enclave, but as we see growth in system calls and increased memory utilization, we observe that its overhead approaches close to SCONE hardware mode.

It is concluded that the growth pattern of overheads in SCONE hardware mode and Intel SDK hardware mode is same for increasing the complexity of the workload. SCONE simulation mode has almost constant overhead as compared to baseline C++ code. For unrealistically small workloads the ratio of SCONE and Intel SDK is highest,

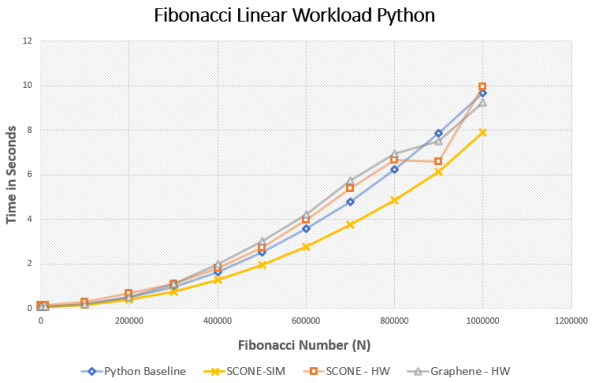

Figure 6.3 – Fibonnaci linear workload in python

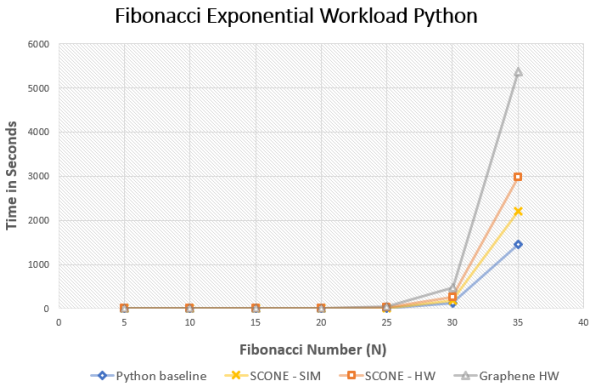

Figure 6.4 – Fibonacci exponential workload in python

but as the complexity of the workload increases, due to efficient handling of system calls and memory management in SCONE its performance matches that of Intel SDK.

- **Experiment: SCONE comparison with Graphene**

In this experiment the performance of SCONE worker is compared with Graphene for python implementation of Fibonacci Linear and Exponential workloads and their respective latencies and overheads are presented. The benchmarking was done for python baseline, SCONE simulation mode, SCONE hardware mode and Graphene hardware mode. For this experiment, SCONE configuration with 2 enclave threads and 6 system threads with increased system and enclave thread spins (SSPINS and ESPINS) was used, as it was a sequential workload with exponentially increasing system calls. For Graphene the Avalon's default configuration with 8 enclave threads was used.

Figure 6.3 shows the latencies of SCONE hardware, SCONE simulation and Graphene hardware modes with respect to native python execution for Fibonacci linear workload. As expected, linear growth of response time is observed in all systems as the number (N) increases. It is observed that the SCONE simulation mode has even better performance than python native execution as it provides better system call handling mechanism. The performance of SCONE hardware mode is better as compared to Graphene hardware mode and is closer to python native baseline.

In Figure 6.4 the comparison of these systems in case of exponential workload is presented. The exponential workload also shows similar results. As the complexity of the workload increase, we observe the latency of SCONE simulation mode increases than the latency of baseline. There is a clear distinction between performance of Graphene and SCONE hardware modes for exponential workload. SCONE shows overall better performance than Graphene in both implementations.

Figure 6.5 shows the comparisons of overhead for SCONE SIM, SCONE HW, Graphene HW as compared to python baseline. We observe that the SCONE simulation mode is more efficient than native implementation for smaller workloads. As the complexity of workload increases the performance of SCONE simulation mode decreases but remains close to baseline and less than hardware mode. The hardware mode of SCONE and Graphene are close to native implementation for smaller workloads but as the complexity of the workload increase, we observe that the gap between SCONE HW and Graphene HW widens. It is concluded that for the configuration stated above, the overhead of SCONE in hardware mode is less than Graphene. For the largest workload the overhead of Graphene is almost twice as that of SCONE.

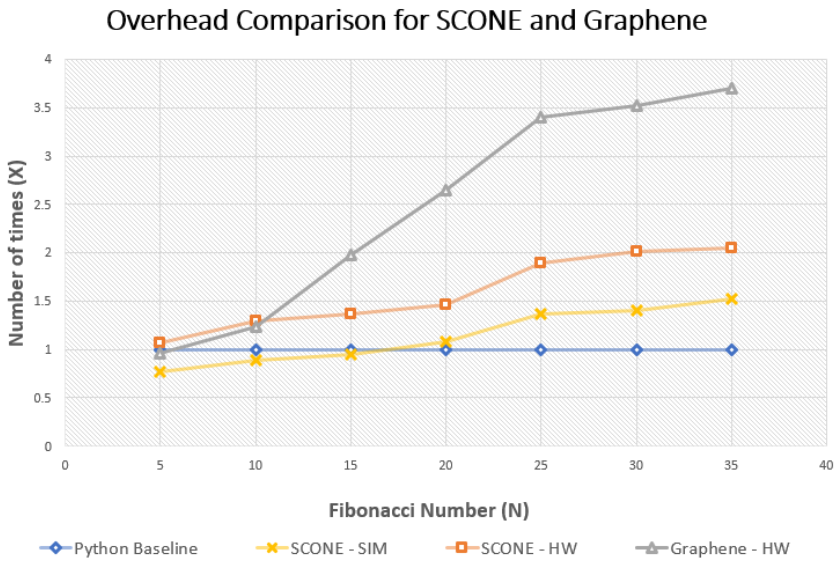

Figure 6.5 – Comparison of overheads in SCONE and Graphene

2. **Performance benchmarking of various SCONE modes**

In these experiments we aim to measure the performance of various modes of SCONE based workers in Hyperledger Avalon. Experiments were conducted on SCONE simulation mode, SCONE hardware mode and SCONE hardware mode with end-to-end file system encryption enabled by CAS. The performance of these modes is compared with the python baseline to observe the overhead of various SCONE modes. For these experiments the default configuration of enclave and system level threads were used i.e., 4 enclave threads and 8 system threads. The versions of the system modules used are SCONE-5.0, CAS-4.2.1 and LAS-4.2.1.

- **Experiment: Fibonacci Linear Workload**
  In this experiment Fibonacci linear workload has been benchmarked on SCONE SIM, HW and HW-CAS modes. It can be observed in Figure 6.6 that all modes of SCONE have greater response time as compared to python baseline, except for SCONE simulation mode. The performance of simulation mode shows similar pattern as

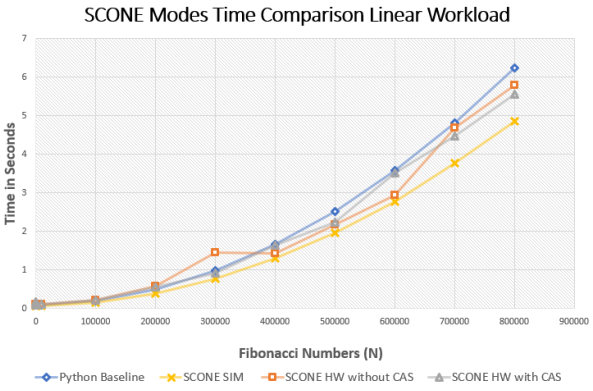

Figure 6.6 – Time comparison of scone modes with linear workloads

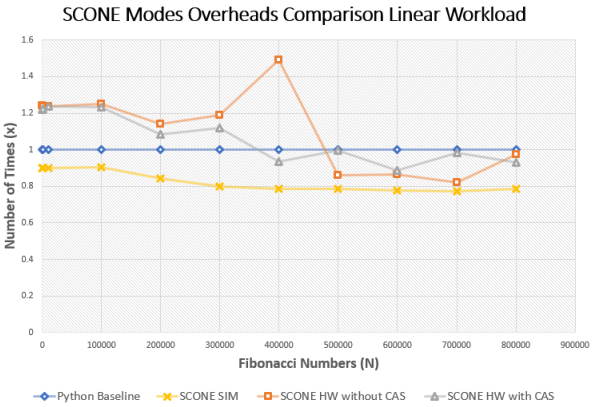

Figure 6.7 – Overhead comparison of scone modes with linear workload

baseline as the workload size grows but it is optimized as discussed in above section. The SCONE hardware mode with CAS shows some spikes but generally follows the same trend until the workload increases. It is shown in Figure 6.7 that the performance of SCONE hardware mode with CAS and without CAS show similar trends and their overheads are close to native baseline for smaller workloads.

- **Experiment: Fibonacci Exponential Workload**
  In this experiment Fibonacci exponential workload has been benchmarked on SCONE SIM, HW and HW-CAS modes. It can be observed in Figure 6.8 that all modes of SCONE have higher response time than python baseline. The performance of SCONE simulation is less than hardware mode. However, the SCONE hardware mode when run in context of CAS session with protected file system shows huge performance degradation as compared to simple hardware and simulation mode. In Figure 6.9 it can be observed that as the complexity of workload increases the overhead of SCONE hardware with CAS mode grows at a linear rate as compared to the other SCONE modes, which exhibit relatively flatter overhead rate. This behaviour could be explained by the exponential increase in file system operations of the workload. As the enclave constantly interacts with CAS for each update in file system the overhead increases inevitably for increasing 'N'.

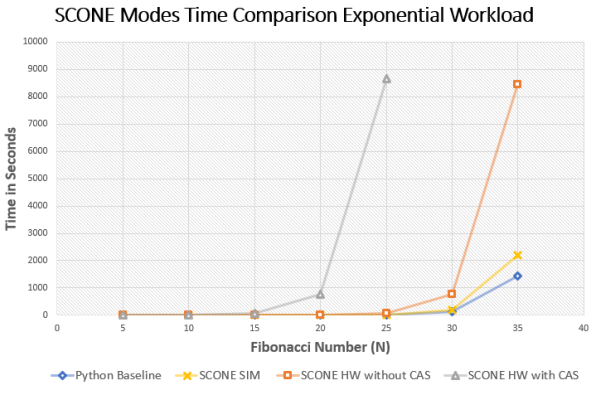

Figure 6.8 – Time comparison of scone modes with exponential workloads

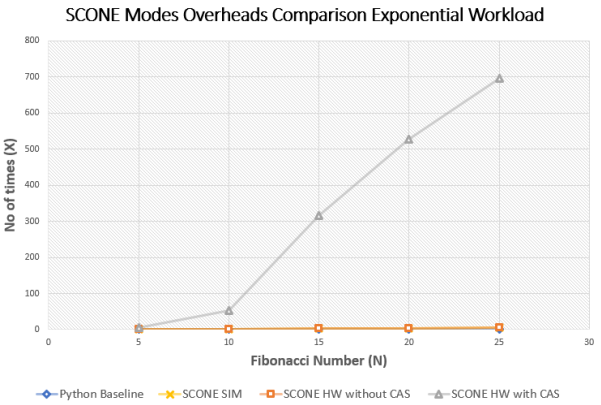

Figure 6.9 – Overhead comparison of scone modes with exponential workloads

3. **Performance benchmarking of scalability of SCONE**
   In these experiments we aim to analyse the scalability of SCONE workers when run in a cluster with Hyperledger Avalon. Experiments were conducted by increasing the number of workers in a cluster from 1 to 5. The performance of increasing number of workers in cluster is compared to the singleton worker and impact of scalability on performance is analyzed.

   - **Experiment: Fibonacci Workload**
     In this experiment Fibonacci linear workload with increasing complexity has been benchmarked for 1 to 5 workers. It can be observed in Figure 6.10 that the response time of workers increase as the number of workers are increased in the cluster. Adding each worker in the cluster has a cost attributed to it as the response time is increased for the same workload across the cluster. This is behaviour is exhibited because each worker is running inside a trusted enclave at same hardware with limited EPC i.e. 94 MB. Currently these experiments have been run on same machine due to non-availability of multiple machines. However, the worker pool can be set up in a distributed cluster for better performance.

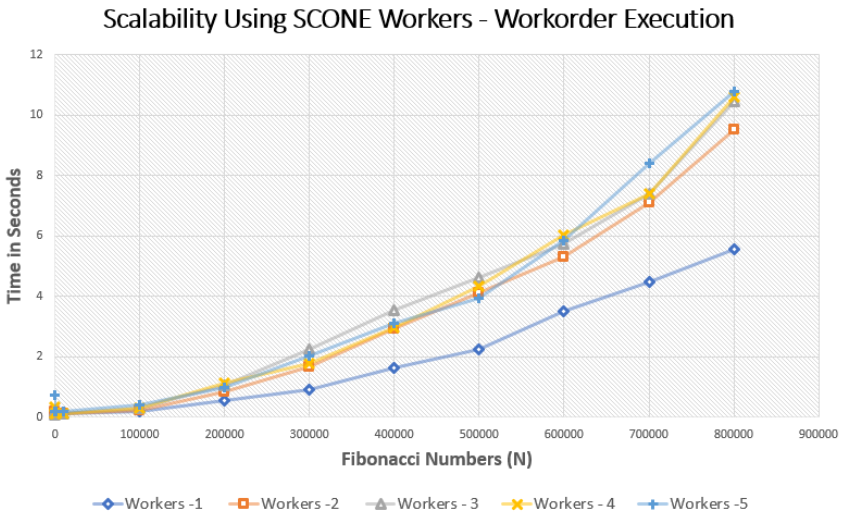

Figure 6.10 – Scalability in SCONE Workers - Workorder Execution

- **Experiment: Scalability impact on bootup**
  In this experiment bootup time of SCONE workers is measured in a pool of workers. We observe in Figure 6.11 that boot time of workers increase at a linear rate as the number of workers increase in the pool. This behavior is also attributed to the sharing of resource among the pool of workers.

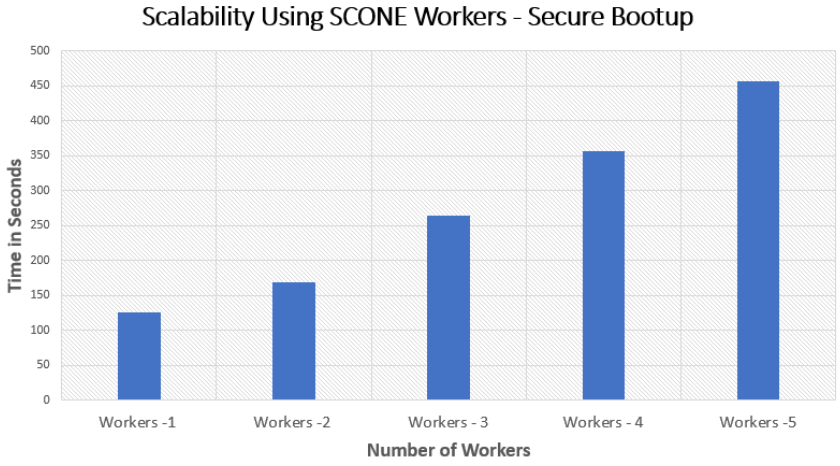

Figure 6.11 – Scalability - Secure Bootup

- **Experiment: Speedup**
  Avalon workers process the requests sequentially by picking them one by one from the queue. If a worker is busy processing the requests, the new request cannot be processed and would increase the response time. Hence the requests can be divided in a pool of workers so that the request can be forwarded to one of the free workers. If multiple workers are running even on the same hardware, speedup can be achieved by parallel processing of request, like threads. Although there is a slight performance degradation by workers sharing the hardware, the overall speedup can make the

system faster. Figure 6.12 shows the speedup achieved by running Fibonacci linear workload for workers increasing from 1 to 5. The maximum speedup of 2.6x was achieved by running 5 workers on the same machine.

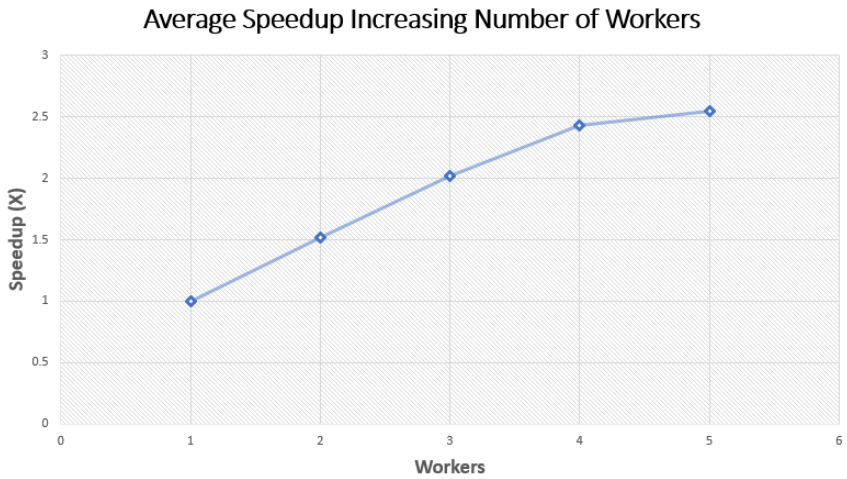

Figure 6.12 – Average speedup while increasing number of workers

7 Future Work

Hyperledger Avalon is a work in progress. It is evolving into many directions and Hyperledger foundation has published a roadmap [Yar] for its future releases. Currently, we have added SCONE based workers in a pre-release version of Avalon. However, the presented solution must be re-evaluated after the production release of Avalon is launched. In this section we discuss further improvements in the design, future implementations and research goals that stem from this work.

7.1 CAS Enabled Design Improvement

In our implementation we use SCONE KME to securely generate keypairs for SCONE workers. There are two keypairs are used for the request response workflow in Avalon.

1. RSA for encryption and decryption
2. ECDSA SECP256K1 for signing and verification

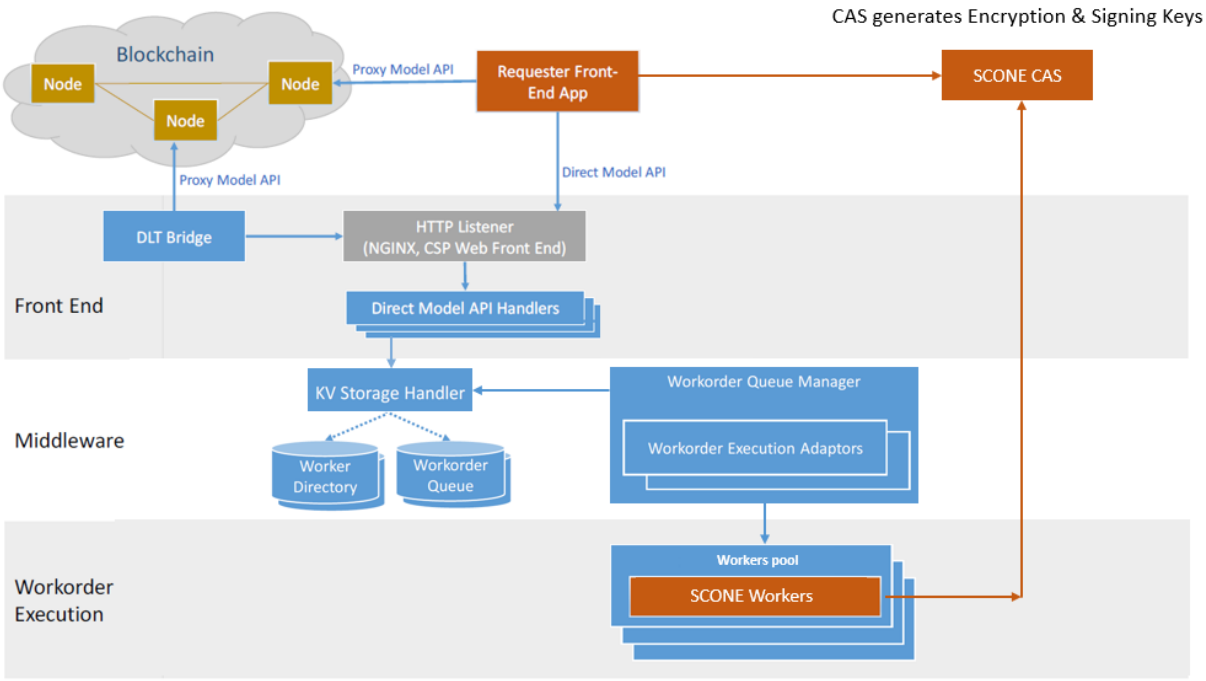

Figure 7.1 – CAS Enabled Future Design Improvement

CAS, by its design has the capability to generate keypairs and secrets for SCONE containers. However, these particular versions of keys are not generated by CAS. So, we use SCONE KME to

securely generate these keypairs and publish on CAS.

If in future, CAS generates the above mentioned keypairs supported by Avalon, then we can simplify the architecture as shown in 7.1

This way, we can remove the SCONE KME (in certain scenarios) and reduce one step from bootstrapping and workers attestation. It will reduce the constant roundtrip time taken for the SCONE KME API call, which can significantly improve the performance.

7.2 Kubernetes based clusters

The current system supports worker pools and orchestration using Docker Swarm. Apart from Docker Swarm, SCONE also supports orchestration using Kubernetes out of the box and Avalon also plans to add Kubernetes support in future versions [Yar]. However, in this work the orchestration using Kubernetes is not specifically implemented due to time shortage. For production use, orchestration using Kubernetes should also be added.

7.3 Blockchain Interoperability using Avalon

We have discussed that one of the main problems in current blockchains is interoperability. Different blockchains are not interoperable with each other and their data and execution results cannot be integrated with each other. However, due to increasing number of customized blockchains, each targeting a domain specific use case, interoperability is a next step.
Hyperledger Avalon provides 'blockchain connectors' in its DLT bridge, which are written for various blockchains i.e., Ethereum, Fabric etc. These blockchain connectors are core part of Avalon's functionality i.e. integration of Avalon with Blockchains networks. Using these blockchain connectors and trusted compute workers, an interoperability solution can be designed using Avalon ecosystem.
In an example scenario we assume that, Ethereum blockchain needs to be integrated with Fabric blockchain. The high-level workflow of this proposal could be as follows.

1. Ethereum blockchain client i.e. smart contract submits data to Avalon's proposed interoperability bridge using Ethereum connector.
2. Avalon does some pre-processing in trusted compute workers (SCONE, Graphene, Intel SDK)
3. Forwards the data to Fabric blockchain using Fabric connector, part of interoperability bridge.

8 Conclusion

In this work we have created an ecosystem of trusted compute workers which can be leveraged to execute custom workloads in trusted execution environments. This ecosystem can solve many problems of existing blockchains such as privacy, scalability and interoperability. The smart contract execution can be offloaded into trusted compute workers for confidential execution.

We have forked Hyperledger Avalon at its pre-release version 0.6 and enabled it to support SCONE based trusted compute workers. SCONE supports execution of legacy applications in docker based containers without any code change or instrumentation. We have also used SCONE's transparent attestation and key management techniques to improve the current design of Hyperledger Avalon. In the end we have evaluated the performance of Intel SDK, Graphene and SCONE based trusted compute workers in the Avalon ecosystem. We found the performance of SCONE based trusted workers almost twice as better as Graphene workers in Avalon ecosystem for complex workloads.

We have also discussed a future design that can improve the workflow of Avalon with SCONE. A proposal of enabling interoperability in blockchains using this system has also been presented as a future work.

A Appendix

List of Abbreviations

CAS	Configuration and Attestation Service	II
DLT	Distributed Ledger Technology	21
EEA	Ethereum Enterprise Alliance	9
EPC	Enclave Page Cache	13
IaaS	Infrastructure as a Service	7
IAS	Intel Attestation Service	15
KME	Key Management Enclave	25
MEE	Memory Encryption Engine	13
MPC	Multi Party Compute	3
PSP	Platform Security Processor	13
SCONE	Secure Container Environment	4
TCB	Trusted Computing Base	15
TC	Trusted Coordinator	7
TCCP	Trusted Cloud Computing Platform	7
TCEBB	Trusted computation-based ecosystem for blockchains and beyond	3
TEE	Trusted Execution Environments	3
TLS	Transport Layer Security	29
TPM	Trusted Platform Modules	7
TVVM	Trusted Virtual Machine Monitor	7
TXT	Trusted Execution Technology	9
ZKP	Zero Knowledge Proofs	3

List of Figures

List of Tables

Bibliography

[AAM19] Maher Alharby, Amjad Aldweesh, and Aad van Moorsel. *Blockchain-based Smart Contracts: A Systematic Mapping Study of Academic Research (2018)*. June 2019 (cit. on p. 2).

[Alh+15] B. Alhinnawi, Gáspár Incze, Ekhtiar Syed, D. Edel, and M. Priom. "THE SNOWDEN REVELATIONS AND THEIR EFFECTS ON EUROPEAN IT-RELATED DECISIONS AND DECISION-MAKING PROCESSES". In: *Proceedings of the 2015/16 Course on Enterprise Governance and Digital Transformation* (Nov. 2015) (cit. on p. 2).

[Ama] Amazon. "Amazon's NitroEnclaves". In: (). URL: `https://aws.amazon.com/ec2/nitro/nitro-enclaves/` (cit. on p. 9).

[ARM] ARM. *TrustZone*. URL: `https://developer.arm.com/ip-products/security-ip/trustzone` (cit. on p. 13).

[Arn+16] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. "SCONE: Secure Linux Containers with Intel SGX". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 689–703. URL: `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov` (cit. on pp. 4, 15).

[Avaa] Hyperledger Avalon. "GitHub". In: (). URL: `https://github.com/hyperledger/avalon` (cit. on pp. 4, 39).

[Avab] Hyperledger Avalon. "Hyperledger Avalon Whitepaper". In: (). URL: `https://github.com/hyperledger/avalon/blob/master/docs/avalon-arch.pdf?fbclid=IwAR2WvqDKBEcuuzTz6X2IubdTqqb1NZxBJvBiiTyzJxBBcUVIp3ep4D2pmmY` (cit. on pp. 17–19, 21).

[Azu] Microsoft Azure. "Microsoft Azure's Confidential Compute". In: (). URL: `https://azure.microsoft.com/en-us/solutions/confidential-compute/` (cit. on p. 9).

[BA17] R. Barona and E. A. M. Anita. "A survey on data breach challenges in cloud computing security: Issues and threats". In: *2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT)*. 2017, pp. 1–8. DOI: `10.1109/ICCPCT.2017.8074287` (cit. on p. 2).

[BC18] Marcus Brandenburger and Christian Cachin. "Challenges for Combining Smart Contracts with Trusted Computing". In: Oct. 2018, pp. 20–21. DOI: `10.1145/3268935.3268944` (cit. on p. 7).

[Bol18] Elena Boldyreva. "Cambridge Analytica: Ethics And Online Manipulation With Decision-Making Process". In: Dec. 2018, pp. 91–102. DOI: `10.15405/epsbs.2018.12.02.10` (cit. on pp. 2 sq.).

[But] Vitalik Buterin. "Ethereum Whitepaper". In: (). URL: https://whitepaper.io/document/5/ethereum-whitepaper (cit. on p. 11).

[BWS19] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. *Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation*. Aug. 2019 (cit. on p. 13).

[BYZ] Sanjay Bakshi, Yevgeniy (Eugene) Yarmosh, and Lei Zhang. "Enterprise Ethereum Alliance Off-Chain Trusted Compute Specification v1.1". In: (). URL: https://entethalliance.github.io/trusted-computing/spec.html (cit. on p. 9).

[Cor] Intel Corporation. *Intel SGX SDK*. URL: https://01.org/sites/default/files/documentation/intel_sgx_sdk_developer_reference_for_linux_os_pdf.pdf (cit. on pp. 14, 39).

[De +17] Stefano De Angelis, Leonardo Aniello, Federico Lombardi, Andrea Margheri, and V. Sassone. "PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain". In: Jan. 2017 (cit. on p. 11).

[EJN] Steve Ellis, Ari Juels, and Sergey Nazarov. "ChainLink". In: (). URL: https://link.smartcontract.com/whitepaper (cit. on p. 3).

[Fuk19] Twaha Fuko. "Intel Software Guard Extensions (SGX) Explained". In: Jan. 2019 (cit. on pp. 3, 13).

[Ger+16] Arthur Gervais, Ghassan Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. "On the Security and Performance of Proof of Work Blockchains". In: Oct. 2016, pp. 3–16. DOI: 10.1145/2976749.2978341 (cit. on p. 11).

[Has19] Jahid Hasan. "Overview andApplications of Zero Knowledge Proof (ZKP)". In: 8 (Oct. 2019), p. 5 (cit. on p. 3).

[HVH18] Simon Hammond, Courtenay Vaughan, and Clay Hughes. "Evaluating the Intel Skylake Xeon Processor for HPC Workloads". In: July 2018, pp. 342–349. DOI: 10.1109/HPCS.2018.00064 (cit. on p. 13).

[KVC19] Thomas Kitsantas, Athanasios Vazakidis, and Evangelos Chytis. "A Review of Blockchain Technology and Its Applications in the Business Environment". In: July 2019 (cit. on p. 2).

[Nak] Satoshi Nakamoto. "Bitcoin". In: (). URL: https://bitcoin.org/bitcoin.pdf (cit. on p. 2).

[Nga+] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. "TrustZone explained". In: (). URL: https://ieeexplore.ieee.org/document/7809736 (cit. on p. 13).

[Ngu+19] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz. "Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities". In: *IEEE Access* 7 (2019), pp. 85727–85745. DOI: 10.1109/ACCESS.2019.2925010 (cit. on pp. 3, 11).

[Ope] Openstack. "Openstack Trusted Compute Pools". In: (). URL: https://docs.huihoo.com/openstack/archive/admin-guide-cloud/content/trusted-compute-pools.html (cit. on p. 9).

[PS10] Maha Prabu and R. Shanmugalakshmi. "An Overview of Side Channel Attacks and Its Countermeasures using Elliptic Curve Cryptography". In: *International Journal on Computer Science and Engineering* 2 (Aug. 2010) (cit. on p. 2).

[Ros+18] S. Roscher, V. Bonisch, J. Lee, Dennis Zeisberg, and J. Schweflinghaus. "Integrating solutions on IBM Z with Secure Service Container". In: *IBM Journal of Research and Development* PP (Feb. 2018), pp. 1–1. DOI: 10.1147/JRD.2018.2795899 (cit. on p. 13).

[SAB] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It Is, and What It Is Not". In: (). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7345265 (cit. on p. 3).

[SAB15] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not". In: Aug. 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357 (cit. on p. 13).

[Sch16] Matthias Schunter. "Intel Software Guard Extensions: Introduction and Open Research Challenges". In: Oct. 2016, pp. 1–1. DOI: 10.1145/2995306.2995307 (cit. on p. 13).

[SGR09] N. Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. "Towards Trusted Cloud Computing". In: *Proceedings of the 2009 conference on Hot topics in cloud computing (HOTCLOUD)* (Jan. 2009), p. 3 (cit. on p. 7).

[She+10] Changxiang Shen, Huanguo Zhang, Huaimin Wang, Ji Wang, Bo Zhao, Fei Yan, Fajiang Yu, Liqiang Zhang, and Mingdi Xu. "Research on trusted computing and its development". In: *SCIENCE CHINA Information Sciences* 53 (Mar. 2010), pp. 405–433. DOI: 10.1007/s11432-010-0069-x (cit. on p. 3).

[Tea] Hyperledger Avalon Team. "Hyperledger Avalon". In: (). URL: https://www.hyperledger.org/use/avalon (cit. on p. 4).

[tea] scone team. *SCONE*. URL: https://sconedocs.github.io/ (cit. on p. 4).

[Tho19] Jason Thomas. *A Case Study Analysis of the Equifax Data Breach* 1 *A Case Study Analysis of the Equifax Data Breach*. Dec. 2019. DOI: 10.13140/RG.2.2.16468.76161 (cit. on p. 2).

[TPV] Chia-Che Tsai, Donald E. Porter, and Mona Vij. "Graphene SGX". In: (). URL: https://www.cs.unc.edu/~porter/pubs/graphene-sgx.pdf (cit. on p. 4).

[TPV17] Chia-che Tsai, Donald E. Porter, and Mona Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX". In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, July 2017, pp. 645–658. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai (cit. on pp. 14, 39).

[Tra+18] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. "ShieldBox: Secure Middleboxes Using Shielded Execution". In: *Proceedings of the Symposium on SDN Research*. SOSR '18. Los Angeles, CA, USA: ACM, 2018, 2:1–2:14. DOI: 10.1145/3185467.3185469. URL: http://doi.acm.org/10.1145/3185467.3185469 (cit. on p. 4).

[Yar] Eugene (Yevgeniy) Yarmosh. "Hyperledger Avalon 2020 Roadmap". In: (). URL: https://wiki.hyperledger.org/display/avalon/2020-04-21+Avalon+Roadmap?fbclid=IwAR3agFMBC2xUVknTSoqOIbo6S7kCbhSo-LfJofugbP-3xlH-VLNZbF6j84A&preview=%2F31199629%2F31200287%2FAvalon2020RoadmapTechForum.pdf (cit. on pp. 4, 47 sq.).

[Yua+18] Rui Yuan, Yu-Bin Xia, Hai-Bo Chen, Bin-Yu Zang, and Jan Xie. “ShadowEth: Private Smart Contract on Public Blockchain”. In: *Journal of Computer Science and Technology* 33 (May 2018), pp. 542–556. DOI: 10.1007/s11390-018-1839-y (cit. on p. 8).