Whitepaper

Hyperledger Avalon with SCONE

written by

Mujtaba Idrees

T-Systems Multimedia Solutions

Reviewers:
Dr.-Ing Ivan Gudymenko
Dr.-Ing. Do Le Quoc
Hira Siddiqui

Date: January 28, 2021

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 and scalability.

Hyperledger Avalon presents a neoteric approach to solve the above-mentioned problems of existing blockchains. It enables creation of a trusted computation-based ecosystem; and allows confidential execution of blockchain workloads in trusted execution environments.

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. Integrity and confidentiality of an execution can be established by the means of trusted computing. One way to provide trusted computing is the use of trusted execution environments (TEE). Trusted execution environment is a reserved area 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.

1.1 State of the art

Hyperledger Avalon [Tea] is a project incubated at Hyperledger foundation that enables creation of an ecosystem of trusted compute workers. It enables lightweight and secure future blockchains by allowing them to execute their heavy workloads in offchain trusted execution environments. 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. 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 trusted execution environments originates from Intel SGX. Intel SGX is a practical solution that constitutes of specific 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 configuration and attestation service (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.

2 Technical Background

In this section, the background knowledge necessary to develop understanding of the topic and the technologies which are trivial to this work i.e. SCONE and Hyperledger Avalon are discussed.

2.1 SCONE

2.1.1 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.

2.1.2 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 project, SCONE file system shields have been extensively used to protect the code and configuration.

2.1.3 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.

2.1.4 Configuration and Attestation Service (CAS)

Configuration and Attestation service (CAS) is one of the most important component 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

2.2 Hyperledger Avalon

2.2.1 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 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 project, we have forked Hyperledger Avalon at "pre-release version 0.6" and made it compatible with SCONE based TEE workers.

2.2.2 System Overview

Figure 2.1 shows high level system components of trusted compute workers eco-system provided by Avalon. The description of components is given below:

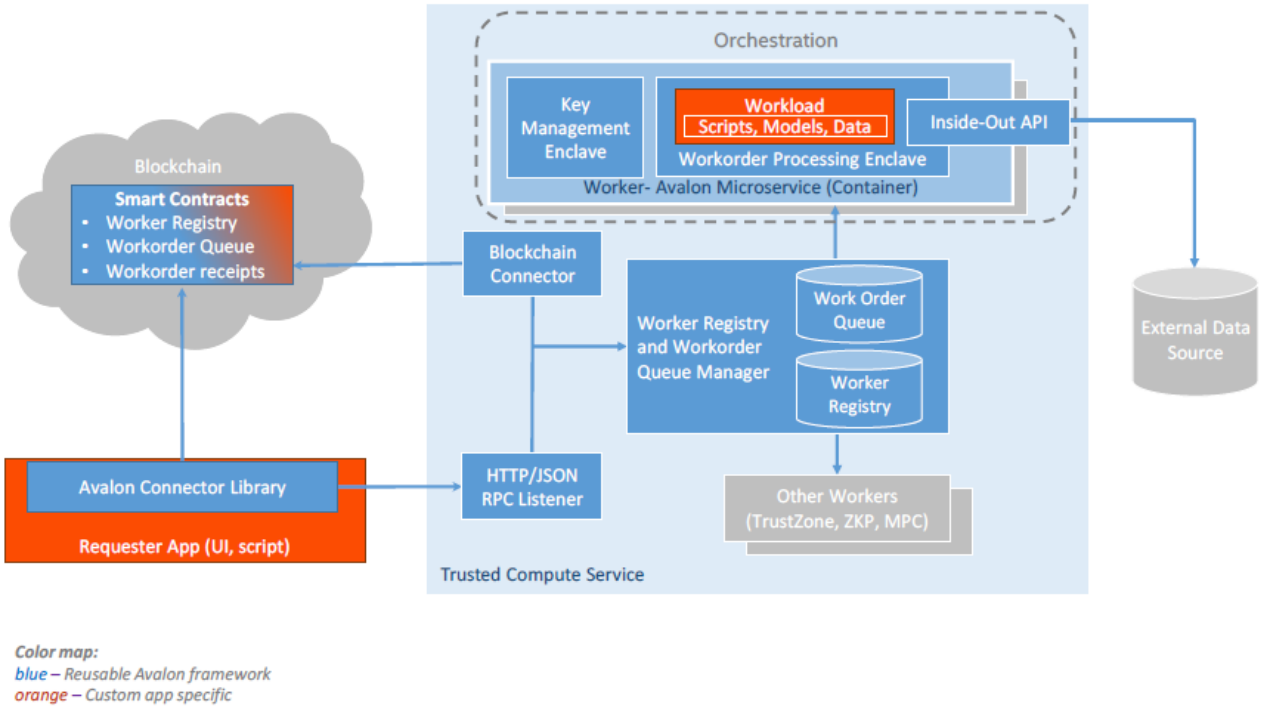

Figure 2.1 – Avalon system components [Avab]

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 Design

The opensource code of Hyperledger Avalon [Avab] is forked to support SCONE [Arn+16] based trusted compute workers in this work. In this chapter, we would discuss the system overview and design goals of the system. Furthermore, we discuss the security aspect and architecture of the system.

3.1 System Overview

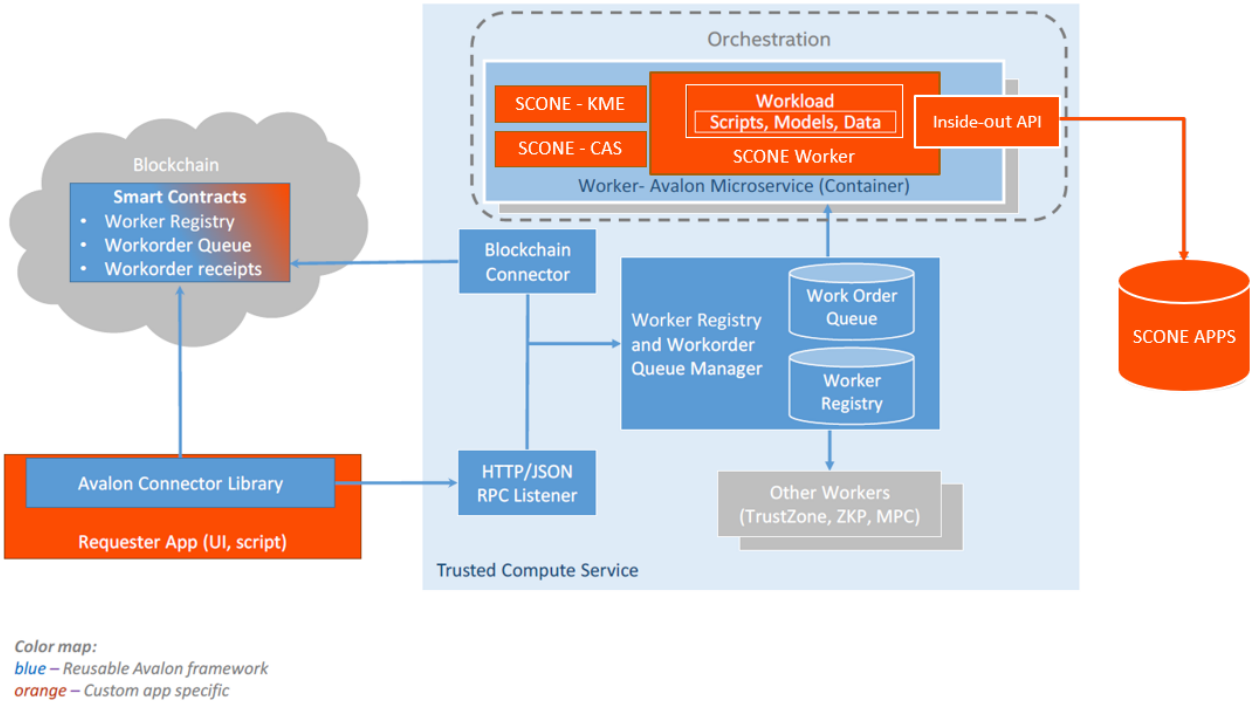

Figure 3.1 – Avalon with SCONE system components

Figure 3.1 shows a high-level overview of the forked Avalon ecosystem. The components of Avalon ecosystem discussed in chapter 2.2.2 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 discussed 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.

3.2 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.

3.3 Security Goals

In presence of an omnipotent adversary, 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.

3.4 System Architecture

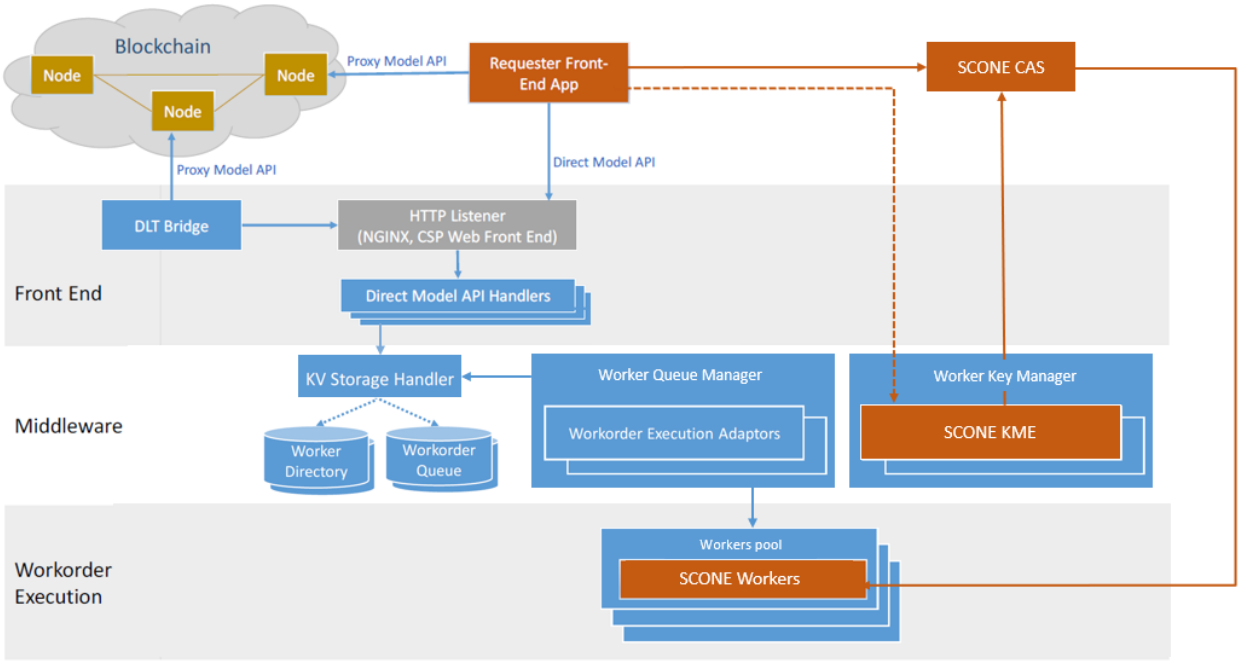

Figure 3.2 – Avalon with SCONE architecture

Figure 3.2 shows the architecture of the forked Avalon ecosystem, that supports integration with SCONE. As discussed in the overview section 3.1 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 **??**. 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.

3.5 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 3.3.

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 the request and response is encrypted.

4 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 project are also discussed.

4.1 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 4.1 – Library versions

4.2 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 3.4. 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 3.4. 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.

4.3 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.

4.4 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. We have provided Fibonacci implementations as example python workload.

   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. In this example openvino SCONE app is used as external application and it has been integrated with Avalon worker using CAS.

      It is a valid real-world usecase considering 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 which runs such complex legacy applications.

   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 Avalon workload and it is integrated with hospital app using CAS. The hospital app now acts as an attested oracle for blockchain.

5 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 that are possible in future.

5.1 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

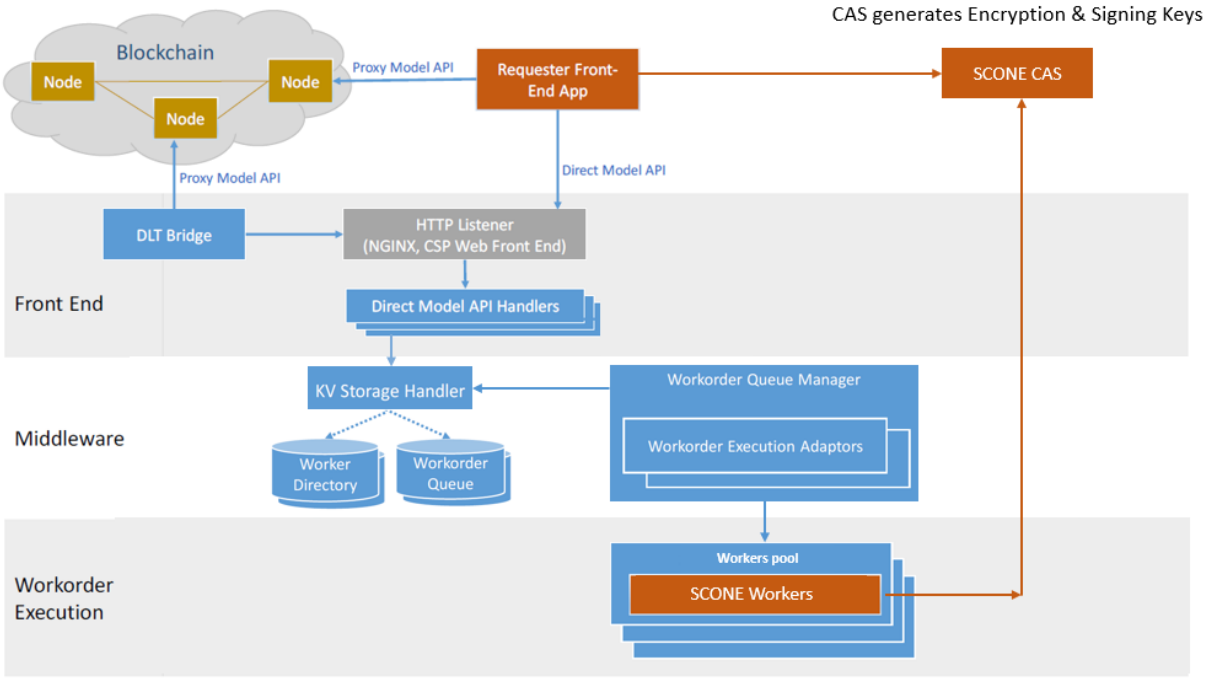

Figure 5.1 – CAS Enabled Future Design Improvement

CAS, by its design has the capability to generate keypairs and secrets for SCONE containers. However, the particular versions of keys that are required by Avalon are not supported 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 5.1

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.

6 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 some problems of existing blockchains such as privacy and scalability. 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 discussed a future design that can improve the workflow of Avalon with SCONE.

List of Abbreviations

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).

[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 sq., 9).

[Avaa] Hyperledger Avalon. "GitHub". In: (). URL: https://github.com/hyperledger/avalon (cit. on p. 3).

[Avab] Hyperledger Avalon. "Hyperledger Avalon Whitepaper". In: (). URL: https://github.com/hyperledger/avalon/blob/master/docs/avalon-arch.pdf?fbclid=IwAR2WvqDKBEcuuzTz6X2IubdTqqb1NZxBJvBiiTyzJxBBcUVIp3ep4D2pmmY (cit. on pp. 7–9).

[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).

[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 p. 2).

[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 p. 3).

[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).

[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 p. 3).

[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).

[Tea] Hyperledger Avalon Team. "Hyperledger Avalon". In: (). URL: https://www.hyperledger.org/use/avalon (cit. on p. 3).

[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).

[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. 3, 21).