Hyperledger Avalon Proposal

*(Originally proposed as **Trusted Compute Framework (TCF)**)*

Project Identifier

HIP: Avalon

Sponsor(s)

Eugene (Yevgeniy) Yarmosh; Intel; yevgeniy.y.yarmosh@intel.com - primary contact

Bowman, Mic; Intel; mic.bowman@intel.com

Middleton, Dan; Intel; dan.middleton@intel.com

Lei ZHANG; iExec Blockchain Tech; lei.zhang@iex.ec

Jean-Charles Cabelguen; iExec Blockchain Tech; jcc@iex.ec

Xiaoning Li; Alibaba Cloud; Xiaoning.li@alibaba-inc.com

Ding, Yu; Baidu; dingyu02@baidu.com

Chen, Zhaofeng; Baidu; chenzhaofeng@baidu.com

Meng Yang; BGI; yangmeng1@genomics.cn

Johann Eid; Chainlink; Johann@chain.link

Andreas Freund; Consensys; andreas.freund@consensys.net

Bill Gleim; Consensys; bill.gleim@consensys.net

Ron Resnick; EEA; ron.resnick@entethalliance.org

Sylwia Rogowicz; Espeo; sylwia.rogowicz@espeo.eu

Ekin Tuna; Espeo; ekin.tuna@espeo.eu

Christopher B Ferris; IBM; chrisfer@us.ibm.com

Gari Singh; IBM; garis@us.ibm.com

Jim Zhang; Kaleido (A ConsenSys Business); jim.zhang@consensys.net

Marley Gray; Microsoft; marleyg@microsoft.com

Przemysaw Simemion; Santander; przemyslaw.siemion@gruposantander.com

Hitarshi Buch; WiPro, hitarshi.buch@wipro.com

Presanna Sundararajan; WiPro; sundararajan@wipro.com

Baohua Yang; Oracle; baohua.yang@oracle.com

Silas Davis; silas@monax.io

Abstract

The Trusted Compute Framework is a ledger independent implementation of the Trusted Compute Specifications published by the Enterprise Ethereum Alliance.

Avalon extends computational trust to off-chain execution enabling.

- Improved blockchain throughput and scalability
- Improved transaction privacy
- Attested Oracles, trusted reporters of data generated outside of the blockchain.

Context

The Avalon prototype has been released to open source as a Hyperledger Lab project at https://github.com/hyperledger-labs/trusted-compute-framework.

Avalon was first developed at another Hyperledger Lab project – Private Data Objects (PDO), which can be found at https://github.com/hyperledger-labs/private-data-objects.

Avalon complies with existing and emerging standards especially, the *Off-Chain Trusted Compute Specification* developed by the Enterprise Ethereum Alliance (EEA). The spec can be found at https://entethalliance.org/wp-content/uploads/2019/05/EEA_Off_Chain_Trusted_Compute_Specification_V1_0.pdf.

A prototype (work in progress) that integrates Hyperledger Avalon and Hyperledger Fabric can be found at https://github.com/jeffgarratt/fabric-prototype/tree/tcf-demo and https://github.com/jeffgarratt/hyperledger-member-summit-2019-tcf-demo-app

Dependent Projects

Hyperledger Avalon does not depend on other Hyperledger projects. However, other Hyperledger projects are encouraged to use Avalon as a component.

Motivation

Hyperledger Avalon enables the secure movement of blockchain processing off the main chain to dedicated computing resources. This enables:

- Improved blockchain throughput and lower latency
- Improved transaction privacy
- Attested Oracles, which are trusted reporters of data generated outside of the blockchain.

Avalon is designed to help developers gain the benefits of computational trust and mitigate its drawbacks. A blockchain is used to enforce execution policies and ensure transaction auditability, while associated off-chain trusted compute resources execute transactions.

Preservation of the integrity of execution and the enforcement of confidentiality guarantees come through the use of a Trusted Compute (TC) option, such as

- Trusted Execution Environments (TEE)
- Multi Party Compute (MPC))
- Zero Knowledge Proofs (ZKP)

The approach will work with any Trusted Compute option that guarantees integrity for code and integrity and confidentiality for data. Our initial implementation uses a Trusted Execution Environment enabled by Intel® Software Guard Extensions (SGX).

Hyperledger Avalon uses a distributed ledger to:

- Maintain a registry of the trusted workers (including their attestation info)
- Provide a mechanism for submitting work orders from a client(s) to a worker
- Preserve a log of work order receipts and acknowledgments

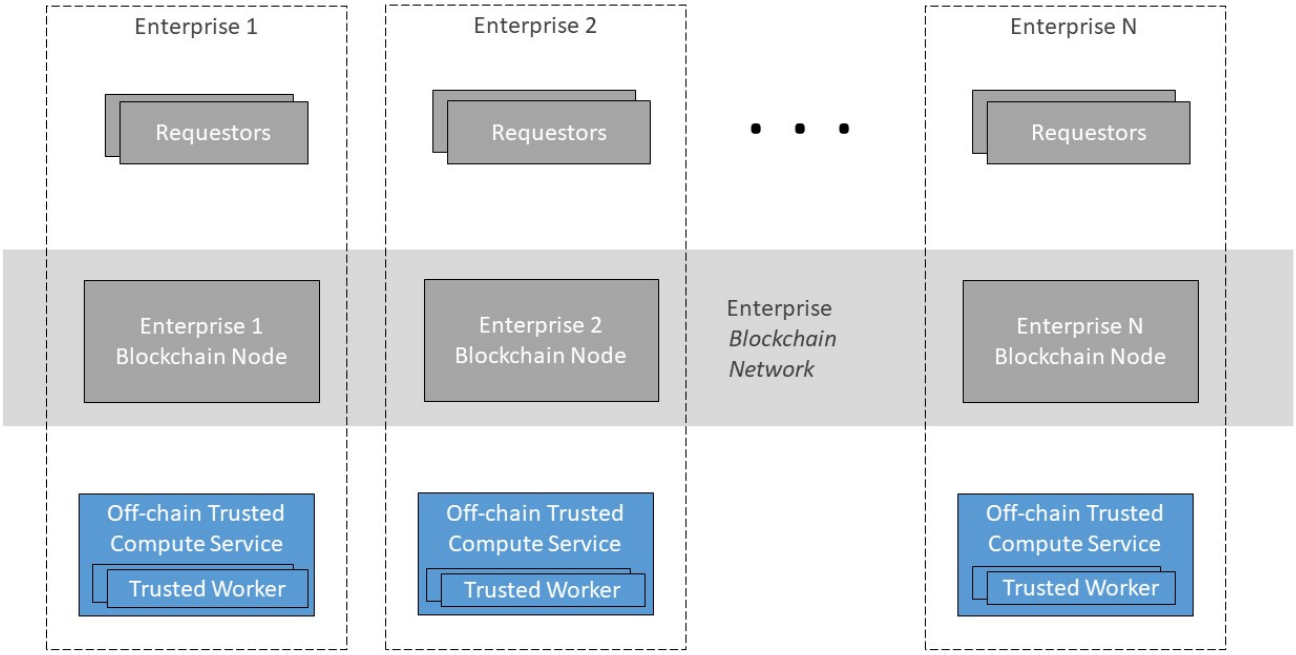
Status

This project started in incubation and is now a full-fledged Hyperledger project.

The initial core functionality of the project has been implemented and the community will deliver additional functionality and bring project quality to product level standards.

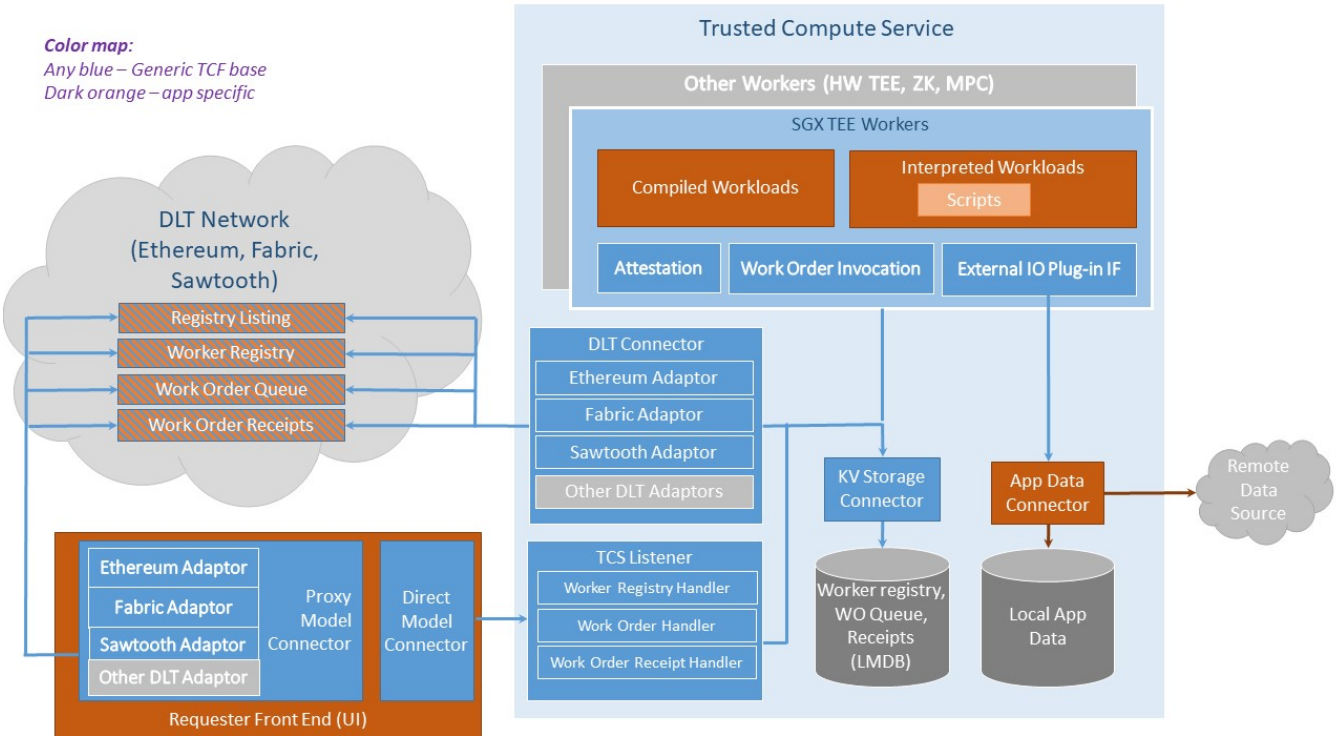
Solution

Early blockchains delivered computational trust via massive replication but had limited throughput, and imperfect privacy and confidentiality. Adding trusted off-chain execution to a blockchain is proposed as way to improve blockchain performance in these areas. A main blockchain maintains a single authoritative instance of the objects, enforces execution policies, and ensures transaction and result auditability, while associated off-chain trusted computing allows greater throughput, increases Work Order integrity, and protects data confidentiality.



The figure above depicts an example of blockchain with N member enterprises. Each enterprise has Requesters, a blockchain node and one or more Trusted Workers (hosted by a Trusted Compute Service). Requesters submit Work Orders, and Workers execute these Work Orders. Work Order receipts can be recorded on the blockchain. While each of the enterprises in the figure above contains all three major components (blockchain node, Requester, and off-chain Trusted Compute Service), this is not necessary. For example, Requesters from Enterprise 1 may send Work Orders to a Worker at Enterprise 2 and vice versa. Ultimately, an enterprise is free to host any combination of the three elements depicted on the figure above. Accessing resources across multiple enterprises increases network resilience, allows more efficient use of resources, and provides access to greater total capacity than most individual enterprises can afford.

A diagram below depicts Avalon architecture at a high-level.



Trusted Compute Service (TCS) hosts trusted Workers and makes them available for execution of Work Orders (WO) that are submitted by Requesters via a front end UI or command line tools. Work orders also can be submitted by (Enterprise app specific) smart contracts running on the DLT.

There are four interfaces implemented according to [EEA-TC-SPEC]. Even though initially designed for Ethereum, these interfaces are uniformly implemented for all supported DLTs. The interfaces are:

- TCS catalog that lists available services and, for each service, provides a blockchain address and/or URI where their corresponding Workers can be discovered
- Worker registry that lists Trusted Compute Workers includes an attestation verification report, public RSA encryption key, and public ECDSA SECP256K1 verification key
- Work order processing API that allows Requesters to submit Work Order requests and receive corresponding responses with data encrypted end-to-end between the Requester and Worker. Both the request and response may include multiple data items independently encrypted by different parties using different keys. The Requester may optionally sign its requests. Even if the Requester doesn't sign the request (aka anonymous request), there is a mechanism enforcing Work Order request integrity. The enclave always signs its responses
- Work Order receipts API that can be used for payment processing, auditing, and dispute resolution. The receipts are signed by the Requester and the worker

There are two models of operation:

- *Proxy model* relies on the smart contract (Ethereum or Sawtooth with Seth) or chaincode (Fabric) running on the DLT for managing all or any subset of APIs listed above. The proxy model has:
    - A DLT connector, which is shown on the diagram above. The diagram depicts components implementing interactions between the DLT and TCS. DLT-specific plug-in adapters are responsible for abstracting DLT-specific APIs from the rest of implementation
    - A Requester, which utilizes a Avalon API running on the blockchain. The Requestor is implemented using a blockchain-specific mechanism, such as Solidity smart contracts on Ethereum or Sawtooth (via Seth) or as chaincode on Fabric.
- *Direct model* provides a JSON RPC API for any of the APIs listed above except for the TCS catalog
    - The TCS listener on the diagram above depicts components implementing the direct model APIs

Direct model was introduced as a complement to the proxy model to facilitate specific use cases that are hard to address by relying on the proxy model alone, e.g. processing sensor data streams (data filtering and pre-processing for IoT or supply chain), handling custom worker key update polices, and aggregating (a large volume of) worker receipts.

Currently, only the direct model is implemented. Proxy model for Ethereum and Fabric will be implemented next.

It is expected that real world applications may benefit from a hybrid model that will combine elements of both proxy and direct models.

A Trusted Worker executes application-specific workloads and implements following capabilities:

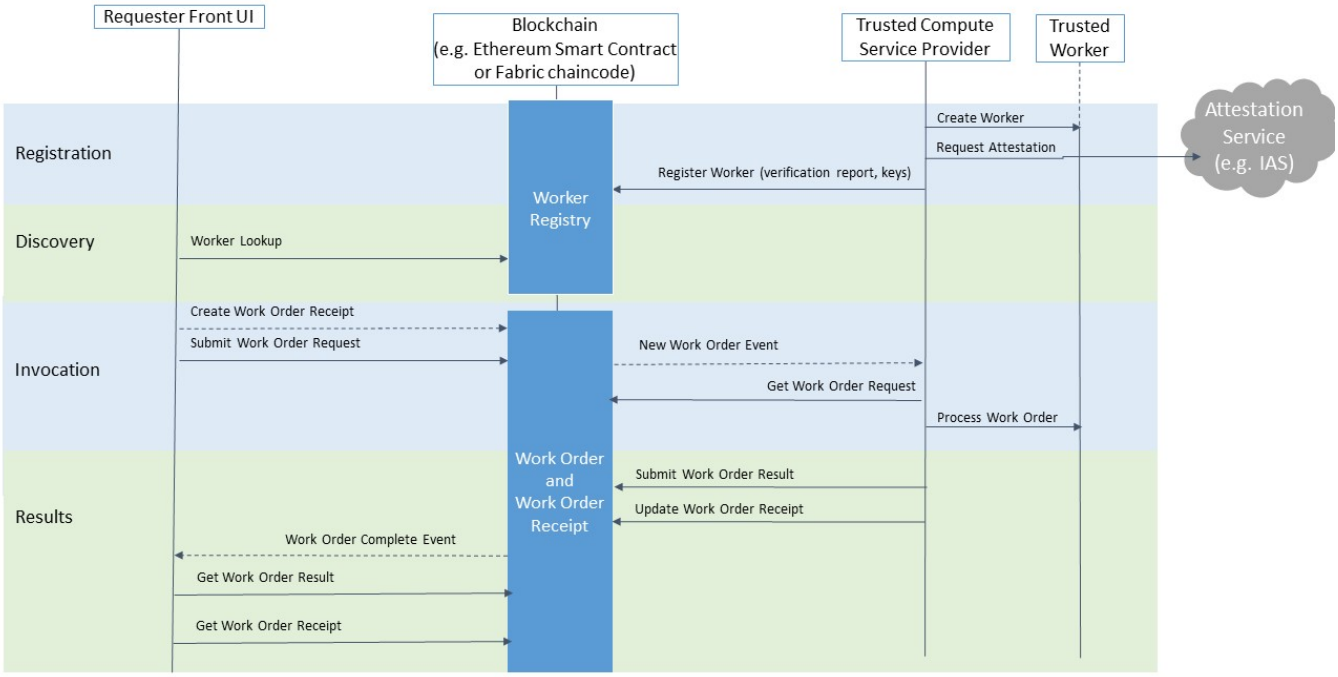
- Worker Attestation service, e.g. in case of Intel SGX TEE it can be an Intel IAS or 3rd party (DCAP) Service
- Work Order Invocation service, which verifies the integrity of the Work Order and the Requester signature, decrypts input and encrypts output data, signs the result, and creates the Work Order receipt update
- External I/O Plug-in Interfaces, which allows a workload running inside of TEE to read and write data from/to external data source, e.g. hosted locally by the TCS or remotely in the cloud. The Interface provides a basic infrastructure to cross trust boundaries. Actual data access and format is application-specific and depicted as Data Connector on the diagram

Workloads can be implemented to execute predefined fixed functions which are pre-compiled at build time (written, for instance in Rust or C++), or can be implemented as runtime script interpreters, (e.g. Solidity or Python). In the latter case the script interpreter is pre-compiled into the worker at build time, and scripts are loaded at runtime as a part of the Work Order. Scripts can be chosen from a list provided by the TCS, or dynamically provided by the Requester (unknown to the TCS). The specific policy enforcement is application-dependent.

In the current implementation, interactions between TCS components is brokered via a KV (Key-Value) Storage that is implemented on top of LMDB. The KV Storage Connector allows utilizing the KV Storage on a single physical system or across multiple physical systems. The Adaptor abstracts LMDB from the rest of TCS so a production implementation may choose to replace LMDB with another database of their choice.

Components shown in blue on the diagram below depict functions that are generic for all or most applications. Components shown in orange depict functions that are always application-specific. Decentralized components (smart contracts or chaincode) running on DLT are depicted as a mix of both colors because in some cases the Avalon baseline implementation is sufficient, while in others dApp may have to differ or extend the baseline implementation.

The diagram below depicts a high level execution flow in the proxy model.



During the registration phase, TCS:

- Instantiates a Worker. The Worker generates signing and encryption key pairs and stores the private keys in a way accessible to the Worker only, e.g. in case of Intel SGX TEE, it will be in its sealed storage
- Produces an attestation quote for the Worker (including an attestation chain for its public keys) and submits it for a verification, e.g. in case of Intel SGX TEE, to Intel IAS
- The attestation verification report from the previous step along with the public keys is submitted to and recorded in the Worker Registry on the blockchain

During the discovery phase a Requester looks up an appropriate Worker and validates its attestation verification report and stores the Worker's public keys for further use.

During the invocation phase:

- The Requester creates a Work Order request payload (JSON). To encrypt the data, the Requester generates a one time symmetric key that also submitted as a part of the request, but encrypted with the Worker's public encryption key. The request includes an encrypted hash of its key parameters and data items (integrity enforcement). The request can also be signed by the Requester
- The Requester submits the request to the Work Order Queue on the blockchain and creates a corresponding Work Order Receipt on the blockchain
- After the Work Order request is submitted to the Work Order queue on the blockchain, TCS receives a notification (aka event or alert) and retrieves the Work Order request from the Work Order Queue on the blockchain. TCS verifies the Requester's signature (if provided)
- The Work Order request is submitted to the Worker. The Worker decrypts the data and verifies the Work Order integrity. Then the Worker processes the Work Order and generates a response. The Worker uses the symmetric data encryption key from the request to encrypt output data and signs the response using its private signing key

During the results phase

- TCS submits the Work Order response to the Work Order Queue on the blockchain and updates the Work Order receipt on the blockchain.
- After the Work Order Result is submitted, the Recipient receives a notification and retrieves the result from the blockchain. It may also retrieve the updated Work Order Receipt. In many use cases the receipt is used by 3rd party (not by a requester itself) for payment processing, dispute resolution, auditing, or regulation compliance
- The requester decrypts the result using the same one-time key that was generated during the invocation phase. The Requester uses the Worker' s public verification key to verify that the result was signed by the Worker (and hence the Work Order indeed was processed by the right Worker).

Efforts and Resources

Initial functional implementation is already available as a Hyperledger Lab project **[TCF-GITHUB]**. The Avalon implementation is derived from another Hyperledger Lab called Private Data Objects (PDO) **[TCF-GITHUB].** Initially a private branch was forked by Intel to build the initial Avalon implementation, with contributions from iExec.

There is a growing Avalon community already. Below is list of companies that have already expressed a formal support for the project. More companies and individuals are in the process of learning and ramping up on Avalon with plans to join the Avalon project and utilize it for their product development. The number of contributors represents the initial commitments, with many companies intending to increase participation as the project advances to the next phase.

- Intel Corporation: 7 contributors to deliver core Avalon infrastructure and Intel SGX-specific code
- iExec: 4 contributors will develop Ethereum smart contracts, integrate TEE options that support most of the the mainstream programming languages and native applications, and improve Avalon easy-of-use for developers
- Alibaba: 2 contributors who will work to adopt Avalon for its Ali Cloud and contribute to the Avalon core to extend supported programming environments, e.g. GOLANG
- Baidu: 2 contributors who will work on enhancing core capabilities and integration of MesaTEE based workers.
- BGI: Will contribute to the integration of Hyperledger Avalon into Hyperledger Fabric
- Chainlink: 3 contributors that will contribute to the Avalon's plans for how to integrate with decentralized oracles and attested oracles, which will be able to provide both Avalon computations and various on-chain computations enabled by them with secure access to various key API inputs and enterprise/payment event outputs.
- Consensys: 2 contributors to work on the Avalon architecture, documentation, and spec compliance
- EEA: expects to use Avalon as a base for its EEA Off chain TC Specification certification program and cooperate with the Avalon community to drive improvements to the Specification
- Espeo: 1 developer to contribute a monitor tool and help with implementation of Ethereum integration
- IBM: 1 contributor to work on integrating Avalon with Fabric
- Kaleido (A ConsenSys Business): 1 contributor working on deployment and manageability solution for hosting Trusted Compute Service
- Microsoft: to provide Azure resources to run a Avalon test net and contribute to Trusted Token usage implementation
- Santander: 3 contributors to provide reference implementation of critical use cases and incorporate Avalon into Santander's cyber-security policies.
- WiPro: 2 contributors working on design and implementation of resource types (ZKP, MPC etc.)

How To

The project will be managed in a Hyperledger GitHub repository and will follow community norms present in other Hyperledger projects.

We propose creation of the following repository: github.com/hyperledger/trusted-compute-framework

References

[TCF-GITHUB]

TCF Hyperledger Labs repo: https://github.com/hyperledger-labs/trusted-compute-framework

[PDO-GITHUB]

PDO Hyperledger Labs repo: https://github.com/hyperledger-labs/private-data-objects

[EEA-TC-SPEC]

EEA Off-Chain Trusted Compute Specification: https://entethalliance.org/wp-content/uploads/2019/05/EEA_Off_Chain_Trusted_Compute_Specification_V1_0.pdf

[TCF-BLOG]

TCF Introduction BLOG: https://software.intel.com/en-us/articles/ecosystem-support-for-eea-trusted-compute-specification-v10-improves-blockchain-privacy-and

[FABRIC-TCF-POC]

GITHUB repositories for the ongoing POC project that integrates TCF with Hyperledger Fabric:

https://github.com/jeffgarratt/fabric-prototype/tree/tcf-demo and https://github.com/jeffgarratt/hyperledger-member-summit-2019-tcf-demo-app

FAQ

What is the connection between TCF and Gardener? TCF and Gardener discovered each other recently and we definitely agree that there is shared scope between these two projects and both projects are interested in working together in the process of evaluating optimal collaboration options.

Chainlink works with Google–does this have any bearing (except for the Ethereum angle)? Chainlink joined TCF to contribute an implementation and use case for attested decentralized oracles.

Intel SGX vulnerabilities–what bearing will this have on the work? Intel tracks information about vulnerabilities and mitigations on a central website these links: facts-about-side-channel-analysis-and-intel-products and engineering-new-protections-into-hardware. Please refer to these websites for information about specific security vulnerabilities, resources, and responses, including mitigation by product CPU model.

What are the identity bindings between off-chain identities (both Enterprise and TCF identities) and on-chain identities? At this time TCF assumes that identities can be either user defined, blockchain address, or the public signature verification key of the requester. This is an area that can be defined further, e.g. to accommodate Ethereum Decentralized Identifiers (DIDs).

Is there any current work or hooks in TCF that deals with other forms of trusted computation (such as Hardware TEEs, HME, MPC, ZK etc.)? The TCF architecture is specifically designed to accommodate additional trusted compute options (via a KV storage interface). There is a work in the process to locate potential contributors with appropriate domain expertise via the Ethereum Enterprise Alliance (EEA). Also TCF encourages anybody with such expertise to join this project and contribute additional worker types to the project.

What is the relation between TCF and Gardener projects? Gardener and TCF are separate but complementary projects. Gardener may adopt TCF core infrastructure for testing purposes during the TCF implementation phase. In the future TCF will be one of the attested Oracle use cases. Espeo (primary Gardener sponsor) will join TCF and to contribute a monitor tool and help with implementation of Ethereum integration.

Closure

The success of this project can be measured by successful integration of TCF with multiple DLTs and, more importantly, by its broad utilization in real world enterprise-focused use cases emphasizing requirements of scalability and privacy preservation.

Reviewed By

- [x] Arnaud Le Hors
- [x] Baohua Yang
- [x] Binh Nguyen
- [x] Christopher Ferris
- [x] Dan Middleton
- [x] Hart Montgomery
- [x] Kelly Olson
- [x] Mark Wagner
- [x] Mic Bowman
- [x] Nathan George
- [x] Silas Davis