Klave
White Paper

JUNE 2023

klave

White Paper Contents

1.0. Abstract

Trust is essential to all relationships and establishing trust can be an expensive and slow process. It can also be riddled with human errors and inconsistencies, highlighted in many recent scandals where trust is often breached and abused.

'Trust, but verify' as the phrase goes. But what if to remediate, we had to trust less?

Imagine a digital world where you could verify that your data is safe and secure. A world where your interactions with others, companies and institutions did not depend on blind trust and you could verify that the systems you rely upon, worked entirely as intended.

Recent developments in the field of confidential computing and trusted execution environments (TEEs) have paved the way for a new category of cryptographically secure, highly available, trustless systems with unparalleled performance.

A technology that will generalise the principle of falsification-resistant computations and ledgers while guaranteeing integrity and privacy. Revolutionising the way we interact with the world around us.

Join us in designing honesty into the very fabric of the internet.

klave™ | **designing honesty into the very fabric of the internet**

2.0. Introduction

2.0. Introduction

The Klave Network empowers developers to create tamper-proof trustless applications protected by secure hardware and cryptography. Providing the infrastructure for developers to easily develop, deploy and manage trustless applications, bringing honesty by design to the web.

2.1. What is Klave?

Klave is a next-generation Trustless Platform as a Service (TPaaS). An open platform designed to enable integrity and privacy in a world where trust is hard to come by. It provides a reliable and secure infrastructure for individuals and organisations to build and run applications without fear of interference from platform providers or other third parties. Apps are executed on Klave with the assurance that the code is running as intended and that third parties (including platform providers) cannot observe or alter the code during execution. In addition, Klave supports privacy use cases as data is always kept secret (always including from platform providers). Providing technical evidence and cryptographic proof to enable verification every step of the way, systematically.

2.2. Confidential Computing meets Distributed Ledger Technology

Klave is built upon a next-generation Distributed Ledger Technology (DLT) called Secure-Enclave Distributed Ledger Technology (SDLT) that combines the power of confidential computing with DLT. The combination of these technologies achieves information confidentiality and produces proof that the integrity of the process has not been compromised. Klave's SDLT is a high-performance ledger and transactional system, capable of supporting any application and serving a wide range of purposes.

2.3. Confidential Computing as a platform, made simple

The technology behind confidential computing is complicated and not accessible to all developers. To keep developers focused on their critical business code, Klave hides all complexity by deploying applications within secure hardware enclaves, making them distributed by default, and ensuring associated ledgers are always encrypted and tamper-proof. As a bonus, it also enables confidential computing use cases from the get-go.

2.4. An open platform for composable Apps

Klave makes it easy for developers to develop, deploy and manage applications by integrating natively within their workflow and by enabling them to code in the language they love most. In addition, Klave focuses on transparency by being open-source and accompanying the development lifecycle from the gpg commit signature of the developer to the application versioning and packaging.

2.5. A platform honest by design

With Klave, you can attest every step of the way that everything is going as expected. We call that an honesty machine. By leveraging privacy-enhancing technology and advanced cryptography, Klave provides technical evidence and proof of honesty systematically.

3.0. Concept behind Klave

3.1. Confidential Computing

Confidential Computing offers an additional security layer to protect business logic and preserve the privacy and integrity of data while it's being processed.

Disk encryption is a technique to keep data private while at rest, for example, when it's stored on a backup storage device.

Network protocols which encrypt data are commonly used to protect data while being transferred between remote machines, preventing attackers from snooping on network connections or internet routers from accessing or tampering with it.

However, once data arrives at its destination and is decrypted to be processed, it can be observed and even tampered with.

High-privileged users, people with physical access to the servers or attackers who manage to gain equivalent access can see, copy, or modify data and even tamper with the results of the computations being performed.

Confidential computing brings new ways to protect data while in use.

Data and computations can be protected using hardware-based TEEs, relying on the traditional techniques for protecting it at rest and in transit, as well as introducing new techniques to deal with potential attacks from the elements that were implicitly trusted before, but no longer need to be.

With traditional computing, the entire software stack such as eventual services from cloud providers, BIOS and firmware, the host operating system and/or hypervisor and guest operating system, to the actual application doing the business logic and all the libraries it depends on, need to be trusted.

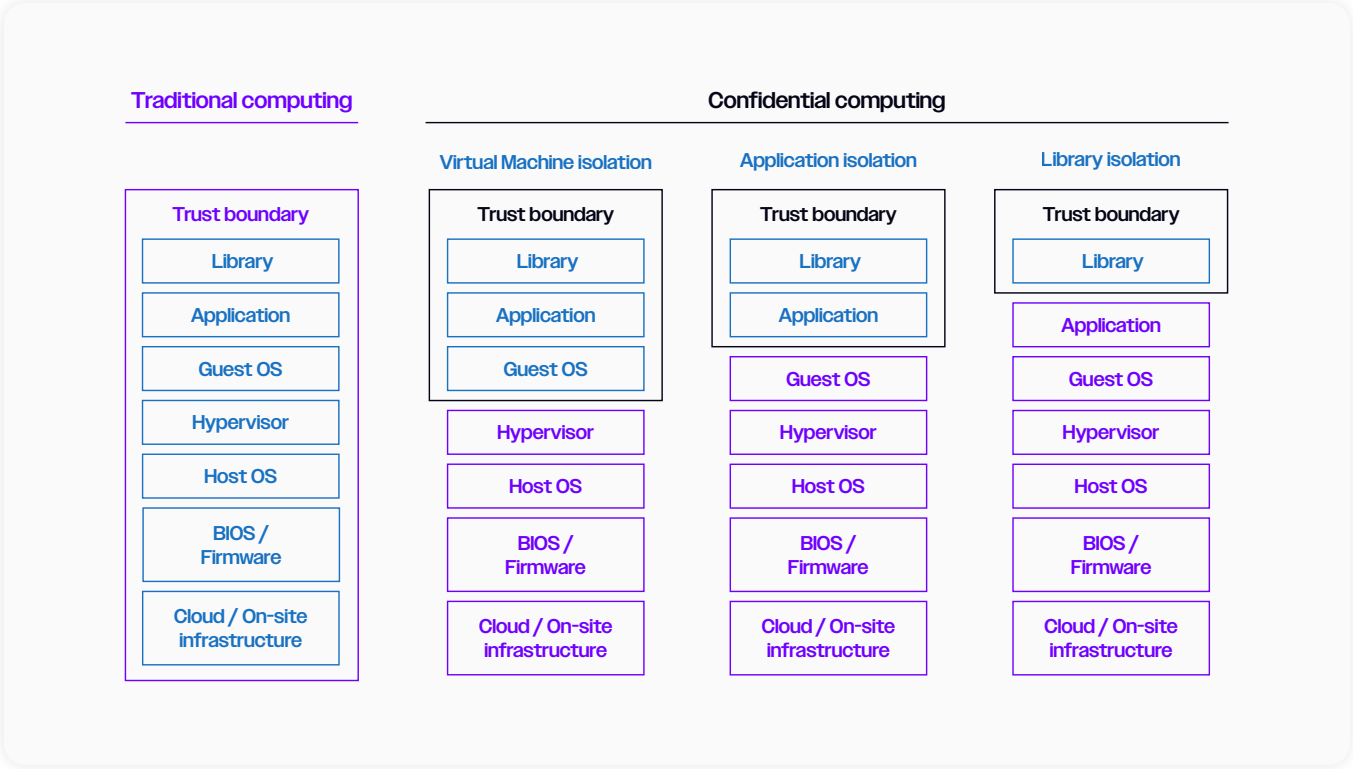

Fig. 1 - Comparison of trust boundaries between traditional computing and several confidential computing isolation scopes.

Confidential Computing introduces different scopes of isolation to reduce the number of layers which can access private data.

With **Virtual Machine isolation**, only the guest operating system, application and libraries have access to private data. The virtual machine runs on the TEE preventing access from the hypervisor or host operating system. An attacker compromising the host operating system won't have access to private data, but one who compromises the guest operating system and escalates privileges will.

Application isolation reduces the trust boundary further, so the operating system doesn't have access to private data. Only the application and its dependencies are running on the TEE and therefore an attacker would need to compromise the application logic to gain access to private data.

Library isolation reduces the trust boundary even further by running only the libraries processing private data on the TEE. The part of the application that doesn't process the private data doesn't need to be trusted but may remain responsible for other tasks such as forwarding messages in and out of the TEE.

Klave leverages on the library isolation model to reduce the trust boundaries to their minimum. Confidential Computing is a huge paradigm shift. When using attestable TEEs it gives the data owners proof of how their data is going to be used before they submit it and guarantees the integrity of the computations' results.

3.2. Trusted Execution Environment (TEE)

In Klave, to keep data and logic protected, Confidential Computing relies on Trusted Execution Environments (TEEs) capable of providing the following guarantees:

- Data integrity
- Data confidentiality
- Code integrity
- Code confidentiality
- Attestability
- Programmability
- Recoverability

The TEE will ensure the logic and data being computed cannot normally be accessed by anyone nor can it be tampered with.

Data going in and out of the TEE is end-to-end encrypted, from the data owner's system to the library responsible for computing it.

The logic loaded onto the TEE cannot be tampered with, meaning that the data will only be used according to the algorithms within the library.

The library loaded onto the TEE is often referred to as an enclave.

On hardware-based TEEs, as is the case of Intel® SGX. CPUs and enclaves are loaded into **Processor Reserved Memory** (PRM) that is encrypted using a key only known to the hardware.

The operating system, highly privileged users or even system administrators with physical access to the hardware cannot normally see the data nor tamper with the logic.

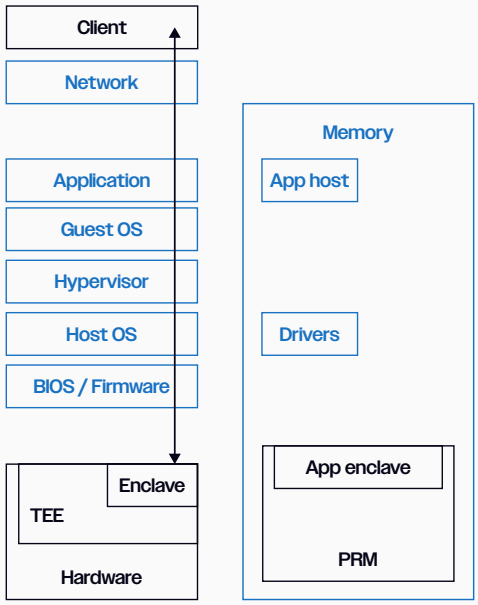

Fig. 2 - Diagram of end-to-end communication between a data owner and an enclave.

TEEs provide proof of which library has been loaded onto it and clients communicating with it can follow the attestation process to verify such proof.

The attestation process also provides evidence that the logic is indeed loaded onto a genuine TEE and whether the platform is up-to-date and contains mitigations for known vulnerabilities.

When vulnerabilities are discovered and mitigated, systems relying on TEEs need to be updated. The mitigation may require firmware upgrades, enclave libraries to be updated or changes to configuration options. This process is often referred to as Trusted Computing Base Recovery (TCB recovery).

3.2.1. Intel® SGX

Klave uses TEEs provided by Intel® SGX. During the manufacturing process of Intel® SGX devices, two keys are fused to the hardware: a Root Provisioning Key (RPK) and a Root Sealing Key (RSK).

- Root Provisioning Key (RPK) - This is a randomly generated key meant to identify each Intel SGX device. Intel is responsible for the safekeeping of all generated keys as they will be part of the attestation process to prove the genuineness of each SGX CPU.
- Root Sealing Key (RSK) - Similar to the RPK, this is a randomly generated key unique per SGX device, but, unlike the RPK, the Root Sealing Key is only known to the CPU.

Intel is responsible for erasing any traces of the generation of this key.

Root keys are only accessible to platform enclaves which can derive keys from them to sign data structures involved in the attestation process.

They are also used for sealing and other key derivations accessible by enclaves.

3.2.2. Enclave

An enclave is a statically linked library containing the logic to process data.

It's dynamically linked to a regular user space process running on unencrypted memory managed by the operating system. The process is responsible for creating the enclave by communicating with the hardware, via runtime libraries and drivers, requesting the CPU to load the enclave onto its protected memory.

3.3. Distributed Ledger Technology

Klave leverages on a next generation Distributed Ledger Technology (DLT) called Secure enclave DLT (SDLT).

A DLT is basically a technological infrastructure around a distributed database and a consensus algorithm allowing simultaneous access, validation, and record updating across all network nodes of a cluster.

The distributed ledger goals are plural:

- Ensuring resilience by distributing a copy of the ledger database across all nodes of a cluster
- Keeping the integrity of the network by recording transactions securely, transparently and being immutable
- Enabling advanced business logic that automatically executes or completes based on prevailing conditions

Klave's SDLT has no central point of control or failure and is therefore more resilient to attacks and less vulnerable to system-wide failure. Leveraging a consensus algorithm based on TEEs capabilities and not on classic consensus based on game theory such as Proof-of-Work or Proof-of-Stake, making it secure against Sybil attack, and also supporting low latency and high throughput enabling transaction instant finality.

Klave's SDLT is used also to remove intermediaries and automate transactions through trustless applications that automatically execute when conditions are met, therefore not requiring a third party or human intervention. Eliminating the need for intermediaries and third parties (including platform providers) is a key characteristic of Klave – reducing costs and increasing efficiency.

4.0. What is a Trustless PaaS?

4.0. What is a Trustless PaaS (TPaas)?

A pioneer of the concept trustless platform as a service (TPaaS), Klave is an innovative variation of the traditional PaaS. It is designed to provide an execution environment for applications that are privacy-enabling, always ensuring computational and data integrity and providing attestation, technical evidence and cryptographic proof to enable verifiability by the user.

4.1. Klave as a TPaaS

In a nutshell, Klave as a TPaaS provides a reliable and secure infrastructure for individuals and organisations to build and run applications without fear of interference from the platform provider or other third parties. Ultimately, aiming to bring accountability back to the business code.

Like classic PaaS providers, Klave allows the deployment of stateful applications with any level of complexity (full Turing completeness). It manages scalability, replication, and the full application lifecycle from commit to deployment.

Klave's infrastructure, built upon cutting-edge hardware is managed by the Klave team.

4.2. Key characteristics of Klave as a TPaaS

- **Data integrity & privacy** - Klave is designed to prioritise data privacy by default and by design. Data is always kept secret and tamper-proof (including from the platform provider). This ensures data integrity and confidentiality at the platform level.
- **Falsification resistance & computational integrity** - Klave prevents tampering with the code deployed by design. Ensuring that the code deployed is what was intended by the user, providing a high level of assurance that the code has not been altered and there is no malicious code injected during execution.
- **Verifiability & honesty** - Klave provides attestation, technical evidence, and cryptographic proof every step of the way and is verifiable by the user. Evidence that the application's code that is deployed is the one intended, attestation that the application is running within a secure hardware enclave, and cryptographic proof that the execution is as intended through transaction and query signature.

5.0. What is a trustless app?

5.0. What is a trustless app?

A trustless app combines an application built on Klave and a front-end user interface. Like open APIs, on Klave public applications are accessible and transparent and can be leveraged by your trustless app.

5.1. Characteristics of a trustless app

Applications deployed on Klave inherit the platform's characteristics and more.

- **Turing complete** - Klave provides a simple Software Development Kit (SDK) to facilitate the development of distributed confidentiality-driven applications (trustless applications) resulting in programs that can be deployed on Klave. These programs can use free-form business logic, including loops and can perform any action given the required resources.
- **Stateful** - Applications deployed on Klave are provided with a private ledger to manage their state. These ledgers allow for complex state management through multiple tables, ensure integrity and stay encrypted at all time.
- **Deterministic** - Given a set of input, transactions of applications deployed on Klave will eventually execute, perform the same function and provide the same outcomes.
- **Confidentiality** - Applications are provided with a unique identity known only by the application runtime. This identity is leveraged on to keep data secret at all time (in transit, at rest and in use) ensuring confidentiality and privacy by design (even from the platform provider). However, Trustless applications are the final frontier in terms of privacy boundary. If needed, data could be exposed by the application.
- **Integrity** - Computational and data integrity are inherited autmatically from Klave. All data stored on Klave are tamper-proof and code execution is falsification resistant thanks to secure hardware and cryptographic primitives.
- **Isolated** - All applications deployed on Klave benefit from two layer of isolation. Klave leverage WebAssembly and each applications run within their own WASM runtime completely isolated from others. In addition, all applications on Klave are deployed with secure hardware enclave running in complete isolation of the OS.
- **Verifiability** - Applications code, deployment, transactions can be analyzed and are guaranteed to execute a predictable ways. In addition, application's ledger structure and secure hardware enclave hosting the application can be attested.
- **Unique identity** - Each Trustless app have a unique identity only known by them. This allows for a unique fingerprint used for secure connection and encryption. Bringing explicit transparency (signature), security (encryption key only known by the app) and integrity to the app.

5.2. Bringing back accountability to the business code

Ultimately, the accountability and privacy boundary are brought back to the application. The platform prevents interference from third parties, however, if the application deployed is extracting the data, they are not secret anymore. Depending on use cases, this will push application developers to follow trustless computing principles and bring transparency by sharing their source code for community inspection and validation. Application developers following these principles could protect themselves from the legal liability of seeing their clients' data. Moreover, they can demonstrate this to their clients and it can be a key driver in many business propositions, even from a strictly commercial point of view.

6.0. Klave design principles

6.0. Klave's design principles

In a rapidly evolving digital landscape, the concept of trust has become increasingly important. Individuals and organisations alike seek platforms that prioritise integrity, confidentiality, and the ability to verify and attest to the actions taken within the system. Klave's design principles are built upon these very needs.

- **Trustless** - At the heart of Klave lies the mission to limit trust in third parties (adversarial or not), including the platform provider itself. This radical approach recognises the potential vulnerabilities and risks associated with relying on external entities and seeks to empower users with greater control over their digital experiences. By reducing the need for trust, the platform fosters a sense of security and transparency, redefining the way we engage with technology.
- **Confidentiality & Integrity** - Klave provides confidentiality and integrity, ensuring that data always remains confidential and unaltered. Advanced cryptography and cutting-edge technologies protect data at rest, in transit, and in use while maintaining the integrity of data and business logic, preventing unauthorised access and tampering. Users can verify that their data and business logic are secure and private, bolstering confidence in the platform's reliability and compliance with regulations.
- **Honesty and Verifiability** - A core principle of this platform is attestability and verifiability. Every action, transaction and operation within the system is designed to be traceable and auditable. Through the innovative implementation of cutting-edge technologies such as TEEs and DLT, users can ensure that the integrity of their data and operations always remains intact. The emphasis on honesty empowers users to validate and verify the actions taken within the platform, cultivating an environment of accountability, without the need for trust.
- **Developer centric** - Klave is developer-centric, prioritising the needs and ease of use for developers. Recognising that developers are the driving force behind the creation of transformative applications and services, the platform provides a comprehensive suite of developer tools, libraries and resources. By simplifying the development process, Klave aims to unlock the full potential of developers, enabling them to create innovative solutions without unnecessary obstacles or complexities..

7.0. Klave architecture

7.0. Klave's architecture

Klave combines the power of confidential computing, DLT and an open development environment to propel trustless applications.

7.1. Isolation

To enable confidential computing, applications are divided into two parts: the untrusted host and the trusted enclave. The untrusted component runs on the unsecured operating system, while the enclave component runs within secure hardware. Enclaves provide secure computations and ensure the protection of secrets. Klave enables the confidential interaction between encrypted data in unsecured operating systems and secure hardware enclaves.

The goal behind Klave's technology is to protect trustless application's business logic and data from untrusted parties or malicious actors (including platform providers) and vice-versa. To do so Klave leverages the best hardware and sandbox isolation.

7.1.1. Hardware isolation

Each trustless app deployed on Klave runs within a TEE called an enclave. Enclaves are an exciting branch of TEEs where a general-purpose chip (typically the main application processor of the system) offers a hardware isolation solution to execute critical code fragments protected from any other system component interference. Klave uses Intel® Software Guard Extensions (Intel® SGX) which offers hardware-based memory encryption that isolates specific application code and data in memory.

Intel® SGX enables falsification resistance capabilities protects against attacks coming from the Operating System (OS), maintains secrecy while processing and provides remote attestation and hardware acceleration for cryptography.

7.1.2. Runtime isolation

The Klave virtual machine hosting the trustless apps is leveraging on WebAssembly (WASM). Each app is compiled in WASM and hosted within a WASM runtime on Klave (WAMR). In simpler terms, on every system which has a WASM virtual machine runtime, a WASM application (binary) will run in the same way. In addition, they all benefit from the WASM sandboxed environment that is separated from the host runtime using fault isolation techniques and ultimately protecting the host from guests.

Mixing both hardware isolation through TEEs and runtime isolation through WASM enables the protection of the guest from the host for the first and the protection of the host from the guest for the second.

7.1.3. Host isolation

As powerful as enclaves are they come with limitations. They live in their own segregated and protected memory area and prevent access to key operating system functionalities. Leveraging only on enclaves would be like having a powerful computer lacking essential components. To remediate and add essential components such as communications, logging infrastructure, and file-system access (database, file management) Klave provides a set of host-side libraries. However, as the host cannot be trusted by enclaves, Klave introduced the concept of crypto-delegation to maintain security between the host and enclaves.

7.1.4. Crypto-delegate

The security model between enclaves and the host is crucial. Enclaves assume the host could be compromised, so they always demand proof of the host's integrity. This is achieved through a crypto-delegate, similar to a zero-knowledge proof. When an enclave calls the host, a cryptographic challenge is included. The host must respond with proof associated with the challenge. Valid proof ensures acceptance of the call, while invalid proof leads to rejection and rescheduling. Klave calls these crypto-delegates, and the concept is called crypto-delegation. They cover key OS functionalities like file-system access, mass storage and environment variables. Klave provides substitutes for system calls, enabling secure sandboxed applications without compromising data integrity or privacy. Crypto-delegation techniques include various cryptographic challenge/proof pairs like reverse hash lookups, merkle proofs and digital signatures.

7.2. Node

A Klave node is a physical server hosting the Klave runtime composed of the host and the enclaves part. The node in addition to running Klave enables you to send, receive or forward information. Klave nodes leverage hardware compatible with Intel® SGX to support the hardware enclave. Each Klave node has its own identity for attestation and direct client connections.

7.3. Cluster and distributed consensus algorithm

To ensure high availability and business continuity, applications are deployed on clusters instead of single nodes. A cluster is a network of nodes communicating through a peer-to-peer protocol. While load-balancing requests (queries or transactions) are a primary function of a cluster, it is equally important to maintain consistency across all nodes to behave as a single system. To achieve this Klave uses the consensus algorithm Raft, which provides an easy implementation and formal safety guarantees.

7.3.1. BFT-Raft

Raft is a consensus algorithm that provides a way to distribute a state machine across a cluster of computing systems, ensuring that each node in the cluster agrees upon the same series of state transitions, through a total ordering of log entries. Raft implements consensus through a leader-based approach, where a leader is elected to manage the replication of the state machine. A Raft leader receives client requests, appends them to a log and replicates them to the other nodes in the cluster. Once the majority of nodes have acknowledged the log entry, the leader commits the entry and notifies the client.

Raft provides a separation of logic that makes it more understandable than other consensus algorithms, such as Paxos, from which Raft is derived. While raft prevents random errors from having a significant effect, it is ineffective against a malicious adversary that tries to subvert its consistency (i.e., not byzantine fault-tolerant).

To make Raft byzantine fault-tolerant, Klave uses the capabilities of TEEs. Implementing features such as enabling mutual attestation between all peers in a cluster, the addition of nodes to Raft consensus and TEEs attestation, signed and encrypted messages between peers and finally the usage of a log journal where entries authenticate the entire history up to the entry.

7.3.2. Logchain

In our version of Raft, we use a log journal (aka. logchain) to store all transactions that are applied to the cluster.

To ensure the security and confidentiality of the data stored in the log, we encrypt the log entries using a strong encryption algorithm. In addition, we protect the encryption keys for the log entries using Shamir's Secret Sharing scheme. Splitting the encryption key for each log entry into multiple shares and distributing each share to different nodes in the cluster, ensures that the log entries are protected even if some nodes in the cluster are compromised. To access the encryption key and decrypt the log entry, a quorum of nodes must come together and combine their shares to reconstruct the key, and therefore it becomes necessary to compromise not one, but several nodes to read the log.

To ensure that a log entry authenticated the entire history up to the entry itself, log entries are chained cryptographically with each log entry containing a hash digest of the previous ones. This ensures the integrity of historical data, similar to the chaining mechanism used in a blockchain but on logs and not on blocks.

7.4. Secure Connection Protocol (SCP)

To connect to an application hosted on a Klave node, users must use the Secure Connection Protocol (SCP). This protocol is designed to authenticate the identity of the remote node owner like any HTTPS would do. But more importantly, SCP allows the end client to verify the identity of the remote enclave running the application and positively identify the code of the enclave.

SCP builds upon TLS, which provides an encrypted channel of communication to the host layer of a Klave node. But additionally, the SCP provides an inner secure communication channel using a 256-bit elliptic curve key pair, where the key pair can be remotely attested to prove that the key has been generated by and protected by a valid TEE and that the TEE is running the correct code. As a result, with SCP there is proof that the server-side program is the correct one, deployed with the integrity and confidentiality properties provided by the TEE.

SCP is a bi-directional authentication, forcing the client to reveal pseudonymous information for enforcing access rights and controls. The SCP is compatible with modern browsers and tablets, making it easy to integrate into web pages and mobile applications to deliver trustless apps.

7.5. Ledger and integrity

One of the key features of Klave is its ability to maintain the state of deployed applications through encrypted ledgers. These ledgers are stored in a key-value storage on the host side but are protected from the host nonetheless. Both keys and values inserted are constantly subject to encryption and cryptographic integrity checks. When an application processes a log, it can result in a combination of multiple actions, such as updating its application state, notifying end users, interacting with other trustless apps and interacting with web services.

The table for keys and values are reverse hash lookup versions that bind a 256-bit hash digest to a key, or a value encrypted using deterministic encryption. The hash is protected from known plaintext and known ciphertext attacks with obfuscation. The table of correspondence binds the hash of the key to the hash of the value and is implemented using a cryptographically authenticated dictionary with a modified Merkle Patricia Trie. Both the key hashes and the value hashes in the authenticated dictionaries are obfuscated to prevent any form of inference.

Klave ledgers are NoSQL in nature, and there are no impositions on how they should be structured. Each table is composed of a table for keys, a table for values and a table of correspondence. Ledgers in Klave are organised in a hierarchical way, with lower levels corresponding to tables and higher levels using the table of correspondence to provide an overarching integrity structure. This allows the state of the entire data in a cluster to be boiled down to a single 256-bit hash, which can be used for proofs and cryptographic commitments and can be used to control and demonstrate transactional integrity in hindsight.

7.6. The Klave Network

A cluster is a network of nodes connected and supporting an app or a group of applications that are transitionally synchronised through BFT-RAFT. Nodes on a cluster can communicate with nodes of other clusters by positively identifying them and authenticating themselves. This is done through a version of the SCP client that sits inside the Klave enclaves and is called ICP (for Inter-Cluster Protocol). Future developments will also allow to transitionally bind several clusters together but assume some degree of transactional sharing. The network of all Klave clusters, public and private, is collectively called the ‘Klave network’.

8.0. Verifiability and attestations

Limiting and trying to remove trust means that the systems should be more transparent and honest. If you can verify, you don't have to trust. To achieve that, Klave provides different techniques to the user.

- **Remote attestation** - A very important aspect of Intel® SGX (and more generally TEEs) is attestation. This is a mechanism for a remote user to verify that the application runs on real hardware in up-to-date hardware and software with the expected initial state. Remote attestation ultimately provides the assurance to the user that the remotely executing SGX enclave is identified and secured and that the correct code is executed.
- **Cryptographic proof** - Klave exposes different cryptographic proofs to check that the code deployed is the code intended, that the communication to the app is secured and encrypted (only decipherable by the app itself) and that the integrity of data is maintained through different techniques (Table hashroot verification, Merkle proof etc.)
- **Transparency** - Klave's code will be made open-source and the fingerprint of the released version of enclaves will be provided to limit the trust in the platform provider even more, enabling code source checking and remote attestation to verify the version deployed.

9.0. User Experience

To ensure Klave's widespread adoption, Klave is focused on developers and provides an environment that integrates directly into their workflow and that is GitOps-friendly from the get-go. On Klave, development and application life-cycle management need to be easy for developers, which is why Klave provides different functionalities to help developers in that endeavour.

9.1. Application scaffolding

In Klave, every application that is developed and deployed must be declared within the platform. This declaration includes a unique name, version and identifier, along with all the APIs that the application provides. APIs can be classified as either queries or transactions. Queries are methods that return information without modifying the underlying state. Transactions, on the other hand, modify the underlying state of an application, which becomes persistent. It's important to note that transactions do not involve payment here but are transactional in the sense of database management systems. Klave is providing an application scaffolding tool that can be found here https://npm.io/package/create-on-klave to scaffold applications within minutes.

9.2. Queries and transactions

Queries are read-only methods that retrieve data from the underlying state but do not modify it. They are typically used to obtain information or perform computations based on existing data without changing it. For instance, a query method might return the current balance of a particular account or compute the average price of a certain asset over a given period.

Transactions, on the other hand, are written methods that modify the underlying state, making changes persistent. They can be used to create new assets, transfer them between accounts, update existing records or perform other operations that affect the state of the system. Transactions are subject to different receipt messages that inform the user of their processing state, including whether they have been successfully executed or failed due to some error.

It's important to note that, due to the consensus semantics used by Klave, there may be uncertainty about the time of execution or even the possibility of execution of a transaction. To mitigate this uncertainty, Klave employs a system of pushing the state of transaction scheduling to keep users informed of transactional certainty. This allows users to better understand the state of their transactions and make informed decisions based on that information.

While some functionalities of the SDK are available in both query and transaction modes, others are only applicable to one or the other. For example, it is not possible to write to a ledger using a query method, as this would violate the read-only nature of queries. Similarly, transactional processing requires additional checks and validations to ensure that the proposed changes are valid and authorised, which is not necessary for read-only queries.

This level of declaration ensures that all applications within the Klave ecosystem are transparent and can be accessed by other applications or authenticated end users. Additionally, it enables seamless integration of different applications in the platform, promoting composability and interoperability.

By having a well-defined and structured way to declare and access applications, Klave provides a secure and streamlined way to develop, deploy and manage the lifecycle of an application. This allows developers to focus on building the logic of their applications, while Klave handles the underlying infrastructure.

9.3. Subscriptions

In addition to standard query execution, Klave's SDK allows for a powerful feature known as subscriptions. When a query is executed as a subscription, it creates an event-driven automation that is constantly monitoring changes in the underlying data. This automation triggers the query in push mode whenever a change occurs that could potentially affect the query result.

This is particularly useful in real-time applications such as trading platforms, where market conditions can change rapidly, and users need to be kept up to date with the latest information. By subscribing to a query, users can receive automatic updates as soon as new data becomes available, without needing to manually poll the data themselves.

Subscriptions can be easily managed by the user, with the option to unsubscribe or disconnect at any time. This ensures that users only receive the information they need, when they need it, without being inundated with irrelevant data.

9.4. Functionalities provided by the SDK

The Klave SDK provides a range of powerful capabilities that allow developers to build privacy and integrity-orientated applications with ease. Here are the key capabilities offered as of 2023:

1. **Ledger:** Offers a robust NoSQL database with advanced indexing capabilities to store and manage data. Developers can define custom schemas, manage access control, and perform complex queries.
2. **Subscriptions and notifications:** Developers can create data triggers with subscription mechanisms to monitor changes in data in real time. This allows for immediate updates and notifications to be sent to the relevant parties when data is changed.
3. **Communications:** Provides a secure communication protocol with the end-user device, including real-time interaction.
4. **Cryptography:** Includes built-in support for cryptographic computations within both queries and transactions. It also provides the ability to manage randomness consistently across a cluster.
5. **Web:** Offers the ability to interact with the web. It is possible to call an HTTPS API, negotiate the sending of an email over SMTP, and call an API to send an SMS or a push notification to a mobile phone. This allows developers to integrate with external services and applications. It is important to note that this is an important change from most distributed systems that prevent such operations from being performed due to the intrinsically random nature of web interaction.
6. **Cross-application communications:** Developers can leverage an ecosystem of applications deployed on Klave and can call other applications natively.
7. **Cross-cluster communications:** Developers can create communications between clusters with subscription mechanisms, allowing for seamless collaboration across different environments
8. **Branching:** Provides the ability to abort a transaction, perform nested transactions via calls to other applications, or enqueue queries dependent on the transactional success of an operation.
9. **Upgrades and releases:** Provides tools to manage releases, allowing developers to deploy new versions of their applications with ease.
10. **AI (Artificial Intelligence):** Currently supports a set of AI libraries focusing on model inference. Some of the models supported include SVM, LightGBM and various regressions. However, Klave's ambition is to be able to provide AI model training within secure enclaves soon.

9.5. Supported languages for writing applications

The Klave SDK uses AssemblyScript which is a TypeScript-like language. It is worth noting that on Klave, the SDK is designed to be compatible with all WASM-compatible languages, meaning that developers can use any programming language that can compile to WebAssembly to build applications. This includes popular languages such as Rust, C and Go, among others. The ability to use other WASM-compatible languages is significant because it allows for greater flexibility in development and integration. It is an effort we have made to attract a broader community of developers, to lead to more innovation and growth for the network.

9.6. How to connect to a cluster

Klave provides an SCP that allows end users to connect to a Klave node, authenticate themselves and authenticate the cluster they want to communicate with through a cluster key. The communication is based on TLS-encrypted WebSockets or TCP to provide two-way communications. This allows the end-user to send commands to the platform and for the platform to push messages back to the end-user. The SCP is provided with a connector that encrypts commands and decrypts responses on the fly on the end-users device.

9.7. Deployment and upgrades

Upgrading and redeploying an application is an essential aspect of software development and Klave provides a seamless way to achieve this. When an application needs an upgrade, it can be done through a transaction, which is traceable and auditable. This upgrade transaction allows developers to update their applications and deploy a new version seamlessly.

10.0. Conclusion

Bootstrapping honesty is at the heart of Klave's proposition. Bringing forward a platform generalising the principles of data integrity and falsification-resistant computation with a focus on privacy by consent, by design and by default. Klave brings honesty and allows for a trustless ecosystem by replacing trust as a belief with a system of proof. It solves the seemingly intractable conflicts between privacy, flexibility (Turing Completeness), security and performance (high-throughput and low latency) with a stateful and multiparty solution. Leveraging secure hardware and drawing upon crypto-graphic techniques from blockchain and differential privacy, allows blockchain-grade integrity as a service. One with much greater performance, at a fraction of its energy cost and with embedded privacy and smart contract capabilities. This is what trustless means, not untrustworthy, but rather the replacement of belief by technical evidence, systematically.

Klave focuses on the developer experience and simplifies access to confidential computing technologies are key to supporting the widespread adoption of trustless computing. This will be largely driven by ease of coding, and accessibility to development in the most popular programming languages. Providing a PaaS offering will also enable developer communities to elaborate ideas quickly and validate business impact in a timely manner.

It has the power to redraw the map of digital services, change our world for the better, strengthen our right to privacy, and, as a by-product, create new business opportunities through trustless collaboration.