Confidential Data Store

Table of Contents

- Overview
- Core Functionality
- Architecture
-
    - Engine
-
    - Store
-
    - Transport
- Data Management
-
    - Initialization & Access Control
- Security and Confidentiality

Overview

This document provides the technical specification for the the Confidential Data Store, a privacy-centric networked storage system specifically tailored to enable programmable privacy in SUAPPs.

Core Functionality

1. Confidential Data Input
2. :
3. SUAPPs define the shape and operations performed on user data, which is put into the confidential data store. Users send data to SUAPPs and, ultimately, the confidential data store through confidential compute requests.
4. Confidential Data Transfer
5. :
6. Once Kettles successfully carry out confidential compute requests, the results are disseminated to other Kettles using a dedicated transport protocol.
7. Data Modeling
8. :
9. The confidential data store adopts a straightforward key-value storage paradigm. This allows flexibility in accommodating various data types:
10.
    - Transactions, UserOps, and EIP 712 signed messages.
11.
    - Bundles, simulation results, and intermediate values.
12.
    - Partial and full blocks.
13.
    - Other extensible data entities.
14. Access Control
15. :
16. To enable programmable privacy, only contracts bearing the correct permissions, termed 'peekers', are permitted to get and put data.

Architecture

The Confidential Data Store consists of three main components: Engine, Storage, and Transport. The Engine is the main orchestrating component responsible for managing calls to Storage, as well as sending and receiving synchronization messages over the Transport. The MEVM is able to directly interact with the Confidential Store Engine through precompiles.

Engine

TheConfidentialStoreEngine is the central component in the Confidential Data Store's architecture and is responsible for coordinating data storage operations and data synchronization across the network. Integrated with the MEVM, it facilitates storage interactions via precompiled contracts. The code snippet below outlines its composition in our currentsuave-geth reference implementation:

type ConfidentialStoreEngine struct

{ ctx context . Context cancel context . CancelFunc

storage ConfidentialStorageBackend transportTopic StoreTransportTopic

daSigner DASigner chainSigner ChainSigner

storeUUID uuid . UUID localAddresses map [common . Address] struct { } }

Store

ConfidentialStorageBackend is the interface that a storage backend for the Confidential Storage Engine must implement.

type ConfidentialStorageBackend interface

{ InitRecord (record suave . DataRecord)

error Store (record suave . DataRecord , caller common . Address , key string , value [ ] byte)

(suave . DataRecord ,

error) Retrieve (record suave . DataRecord , caller common . Address , key string)

([ ] byte ,

error) FetchRecordByID (suave . DataId)

(suave . DataRecord ,

error) FetchRecordsByProtocolAndBlock (blockNumber uint64 , namespace string)

[ ] suave . DataRecord Stop ( )

error } Implementation Note: in oursuave-geth reference implementation, we provide twoConfidentialStorageBackend : the LocalConfidentialStore, storing data in memory in a simple dictionary; and RedisStoreBackend, storing data in Redis. Redis was chosen to allow for fast iteration, but is not meant to be a long term solution.

Transport

StoreTransportTopic is the interface that must be implemented by a transport engine for the Confidential Storage Engine.

type StoreTransportTopic interface

{ node . Lifecycle Subscribe ( )

(<- chan DAMessage , context . CancelFunc) Publish (DAMessage) } Implementation Note: in oursuave-geth reference implementation, we provide an implementation using a shared Redis PubSub in RedisPubSubTransport, as well as a crude synchronization protocol. Note that Redis transport only synchronizes the current state; there is no initial synchronization - a newly connected node will not have access to old data for now. This synchronization protocol will be defined in upcoming specifications.

Data Management

Data Management is at the heart of the Confidential Data Store, ensuring data is stored, retrieved, and maintained securely and efficiently.

Initialization & Access Control

The Confidential Data Store ensures that only legitimate entities can store data, and only authorized parties can retrieve it.

- Data Registration
- : When initializing, it's vital to ensure that only valid data is registered in the system. Data must satisfy specific criteria, such as proper formatting, authentication, and non-duplication. Still, these requirements do not restrictwhat
- type of data gets put in, merely that it should follow a high-level format.
- Access Permissions
- : A robust access control mechanism restricts data access. Only contracts with appropriate permissions (peekers) can access stored data. This mechanism is currently context-based, and the specific access control mechanism is laid out in theMEVM

Security and Confidentiality

Ensuring the confidentiality of the stored data is paramount. Current implementations are still works in progress, and data should not be considered secure. Future iterations will delve deeper into encryption methodologies, access controls, and

auditing mechanisms to fortify data privacy further. Edit this page Previous Confidential Computation Next Block Building