Sentinel: A System for Decentralized Data Governance

Nikhil Sharma

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-77
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-77.html

May 17, 2019

UC Berkeley MS Technical Report

Nikhil Sharma

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee

Dawn Song
Research Advisor

(Date)

⋆ ⋆ ⋆ ⋆ ⋆ ⋆ ⋆

Satish Rao
Second Reader

(Date)

This work proposes Sentinel, a new architecture for decentralized data governance. In a world with growing privacy concerns, it becomes increasingly necessary for data management systems to be trust-free to end users. This paper describes and provides empirical results for enabling data analytics over statistical databases containing sensitive data without requiring any centralized trust and without compromising user privacy. Using smart contracts for decentralized verification of data access policies and trusted execution enclaves (TEEs) for secure computation, this architecture describes a scalable solution for achieving this, supporting the goals of data providers and data consumers alike.

Contents

List of Figures

List of Tables

1. Introduction

This section provides high-level background information and the underlying context and motivation for this project.

1.1. Motivation

The ubiquity of cloud computing over the last couple of decades has fundamentally changed the way we think about data. The power of advanced data analytics and machine learning algorithms, when applied to "big data" has almost limitless potential in terms of the insights it can provide. Such insights catalyze further advances in research at the frontier of almost every field of study, yield significant performance optimizations, and generally provide immeasurable societal value.

A large part of this paradigm shift to cloud computing has been maintaining good practices for **data governance**, an organization's means by which to ensure high quality data throughout the data lifecycle. Data governance strives to ensure goals such as data availability, security, integrity, and usability. However, a fundamental limitation of governance is the existence of **data silos** - repositories of data that are under the jurisdiction of a single organization and unusable by entities outside of that organization. Due to the non-transitive nature of trust, siloed data is typically either not shared between organizations or done in a manner that violates the underlying users' privacy, which was the case during Facebook's infamous Cambridge Analytica scandal in 2018.

Deconstructing data silos and enabling cross-organizational data governance while still respecting user privacy is a lofty but worthwhile goal, one which would increase the quality of machine learning models and data insights by nontrivial margins. By building on top of the privacy-preserving and scalable Oasis Labs blockchain, this research project designs and prototypes a platform for achieving this goal via *decentralized* data governance, one which enables trustless privacy and empowers users to retain full control over who can see their data and how it is used while still enabling data-driven applications to thrive.

1.2. Overview

Smart contracts are decentralized programs which enforce conditions between mutually distrusting parties. As a result of security assurances stemming from attributes of blockchains such as replication of computation across multiple compute nodes, once a smart contract has been deployed even its creator cannot undercut its correct expected execution. However, smart contracts and blockchains in general are known for transparency of information, which is not a desired property for data governance where compliance and regulation are extremely important.

As noted in [18], traditional cryptographic solutions such as zero-knowledge proofs [1] and secure multiparty computation [8] require too much performance overhead to be successful in a governance system supporting a sufficient number of users for most typical enterprise use cases. Instead, a better solution is to use trusted execution environments (TEEs), secure hardware which allows execution of code on untrusted remote servers using cryptographic proofs of validation known as attestations and can be optimized significantly for high-throughput tasks such as machine learning and other analytics algorithms as shown in [13, 21].

In this work, we propose SENTINEL, a data governance platform which utilizes a smart contract ecosystem to enforce a rich set of access policies and produce audit logs for data providers, leveraging trusted execution environments to achieve this at scale while preserving confidentiality and privacy. Though SENTINEL is applicable generically, we will focus on medical use cases in particular to support motivating examples in an industry where carefully regulated governance is especially useful and important.

1.3. Contributions

In summary, the contributions from this research project are as follows:

1. A framework for decentralized data governance which guarantees regulated and auditable data usage for data providers, and correctness and completeness of data for data consumers.

2. An analysis of typical use cases and corresponding privacy and performance analysis.

3. An evaluation of key performance metrics for standard governance operations, verifying platform usability and scalability.

2. Preliminaries

This chapter discusses preliminaries in several areas of focus relevant to this research project.

2.1. Smart Contracts and Blockchains

A **blockchain** can be imagined as a decentralized state transition system which tracks events known as **transactions** (Tx) and computes each new state as a function of the current state and most recent transaction:

$$S_{i+1} = f(S_i, Tx_i)$$

Blockchains are decentralized because state transition computations are replicated across many compute nodes, which resolves inconsistencies from errors such as network failures, malicious nodes, etc. in the newly computed state S_{i+1} via a **consensus protocol**. Each block in a blockchain represents a new set of transactions that occurred since the creation of the previous block, and possesses a header which serves as an authenticated data structure for its contents. This block header stores (i) the hash of the previous block, (ii) creation timestamp, (iii) a consensus proof, and (iv) the Merkle root of the tree formed from the block's transactions. **Light nodes** store only these headers, while more heavyweight **full nodes** opt to store the transactions used to generate the Merkle root as well.

Smart contracts are decentralized applications built on top of a blockchain that maintain some internal value, releasing that value only when certain constraints programmed into the contract are satisfied. Decisions about releasing value are once again taken by replicated computation regulated by a consensus protocol.

Under the Ethereum paradigm [22], in order to regulate smart contract execution cost and prevent abuse, transactions are limited by the amount of **gas** they have been supplied. Gas is a measure of the number of computational steps a transaction is allowed to take before being terminated, and so transactions must specify both a `STARTGAS` and `GASPRICE` indicating how much value the initiating participant is willing to spend to complete it. In general, the gas required to complete a transaction correlates strongly with the time to complete that transaction.

Contracts and external users are both considered network participants and each are associated with an **account**. Smart contracts can initiate execution of other contracts via cross-contract function calls, and so a useful abstraction is to consider transactions as value flow between accounts.

2.2. Trusted Execution Environments

Trusted execution environments, otherwise known as **enclaves**, allow for isolated usage of sensitive code and data from all other software in a system. Enclaves such as Intel SGX have become increasingly prevalent in cloud applications, given their formal security guarantees allowing for secure execution within an untrusted hosting application.

Enclaves are initialized with the help of the untrusted application operating system, which allocates a portion of virtual memory within which the enclave can reside. Intel SGX does this by storing sensitive code and data within a data structure known as the Enclave Page Cache (EPC). SGX also includes a memory encryption engine (MEE) which encrypts and authenticates any enclave data that is evicted to memory, ensuring integrity and freshness of anything that passes through it.

Correct resource management is possible via **remote attestation**, allowing remote users to establish trust in an enclave. In Intel SGX, code residing within an enclave can request a **quote** containing cryptographically signed measurements of the enclave's initial state. These signed measurements serve as a proof to the remote user that the enclave has been initialized correctly to its expected state, and no untrusted code outside of the enclave can access its contents. Once this verification process is complete, the remote user can set up a secure channel with the enclave and provision secrets and/or communicate as necessary to acquire the desired computational results.

Overall, a correctly-functioning secure enclave must be able to do both of the following:

1. Isolate itself within an untrusted host environment, with carefully regulated interaction with the outside world.

2. Remotely attest to its own identity, providing a cryptographic proof of correct initialization to the remote user.

These two functions are necessary in order to enable trusted computation on an untrusted platform.

2.3. Differential Privacy

Differential privacy [3] is a constraint which formalizes and enforces the privacy of individual entities when aggregate information is publicly published about a statistical database within which those entities' data is stored. The theory of differential privacy is developed under the assumption that, regardless of whether or not a given entity chooses to include their personal data in any aggregate query, the conclusions derived from that query will be identical.

We define the **probability simplex** $\Delta(B)$ of a discrete set B to be

$$\Delta(B) = \left\{ x \in \mathbb{R}^{|B|} : x_i \geq 0 \text{ for all } i \text{ and } \sum_{i=1}^{|B|} x_i = 1 \right\}$$

the set of all possible probability distributions over B. A randomized algorithm $\mathcal{M}$ with input space A and output space B is associated with a mapping $\mathcal{M} : A \longrightarrow \Delta(B)$. For any $a \in A$, $\mathcal{M}$ selects elements each element of B with probability defined by the corresponding probability distribution selected from $\Delta(B)$.

Representing databases containing sensitive information as data vectors of their histograms $x \in \mathbb{N}^{|\mathcal{X}|}$ where $\mathcal{X}$ is the universe from which data entries are selected, we define the distance between databases as their **Hamming distance**:

$$d(x, y) = |\{i | x_i \neq y_i\}| \;\; x, y \in \mathbb{N}^{|\mathcal{X}|}$$

and say x and y are neighbors if $d(x, y) = 1$. A randomized algorithm $\mathcal{M}$ then preserves $(\epsilon,\ \delta)$-differential privacy if for any set S of possible outputs and neighboring databases $x, y \in \mathbb{N}^{|\mathcal{X}|}$

$$\Pr[\mathcal{M}(x) \in S] \leq e^{\epsilon} \Pr[\mathcal{M}(y) \in S] + \delta$$

Above, ϵ is defined as the **privacy budget** and controls the strength of the privacy guarantee a given algorithm provides. The δ parameter allows for a non-zero probability that the guarantee fails, but moving forward we will typically only consider *pure* differential privacy, the case where $\delta = 0$. Additionally, we define the l_1**-sensitivity** of a function as follows:

Definition 2.3.1 (l_1-sensitivity)**.** The l_1-sensitivity of a function $f : \mathbb{N}^{|\mathcal{X}|} \longrightarrow \mathbb{R}^k$ is

$$\Delta f = \max_{\substack{x, y \in \mathbb{N}^{|\mathcal{X}|} \\ ||x - y||_1 = 1}} ||f(x) - f(y)||_1$$

In other words, the l_1-sensitivity of a function f is the impact a single individual's presence can have on its value for a given input database.

2.4. Machine Learning

Machine learning is a statistical technique for predictive modeling. Formally, a *supervised learning algorithm* is a type of machine learning algorithm that solves the following optimization problem:

$$\min_{\theta} \mathcal{L}(f_\theta(x))$$

where $\mathcal{L}$ is a *loss function* used to measure the accuracy of a predictive function f parametrized by *weights* θ. Supervised learning solves this problem empirically, using a provided set of inputs $X_1, \ldots, X_n$ and corresponding expected outputs $y_1, \ldots, y_n$ to optimize f with respect to the objective above. Once this process is completed, predicted outputs for new inputs can be computed as

$$\hat{y} = f_\theta(X)$$

Supervised learning often appears in the healthcare and pharmaceuticals industry, where patient records can be used for tasks such as predicting disease rates or determining whether a patient is fit for a certain drug or not. However, medical records are one of the many types of data that may come with a privacy constraint. Machine learning and differential privacy can come hand in hand, as both strive towards the goal of generalizing a distribution of data without need for revealing any individual points. If trained properly, a machine learning model contains information on all the points in a dataset without exposing or overly relying on any single data point.

3. System Architecture

This section provides a thorough description of the design of the Sentinel platform.

3.1. Assumptions and Threat Model

We define a network with three classes of participants:

- **Data Providers**: Data providers interact with the platform by uploading datasets. Their goals may include secure data storage, monetization of proprietary data, and/or outsourcing data analysis with policy-driven and auditable data accesses.
- **Data Consumers**: Data consumers interact with the platform by submitting executables for computation over datasets of their choosing. The goal of data consumers is to derive insights from data analytics over previously inaccessible, siloed data.
- **Policy Developers**: Policy developers write smart contracts that encode data access policies, which can be made publicly available for reuse by providers.

Users can identify as one, two, or all three of these roles via their interaction with Sentinel. We assume a threat model with an honest-but-curious platform and shared external producer-consumer database and actively malicious consumers, where providers do not trust consumers beyond the specifications of the policies they choose to enforce. Furthermore, we assume users trust their local machines and browsers but consider side channel attacks out-of-scope.

3.2. Basic Solution

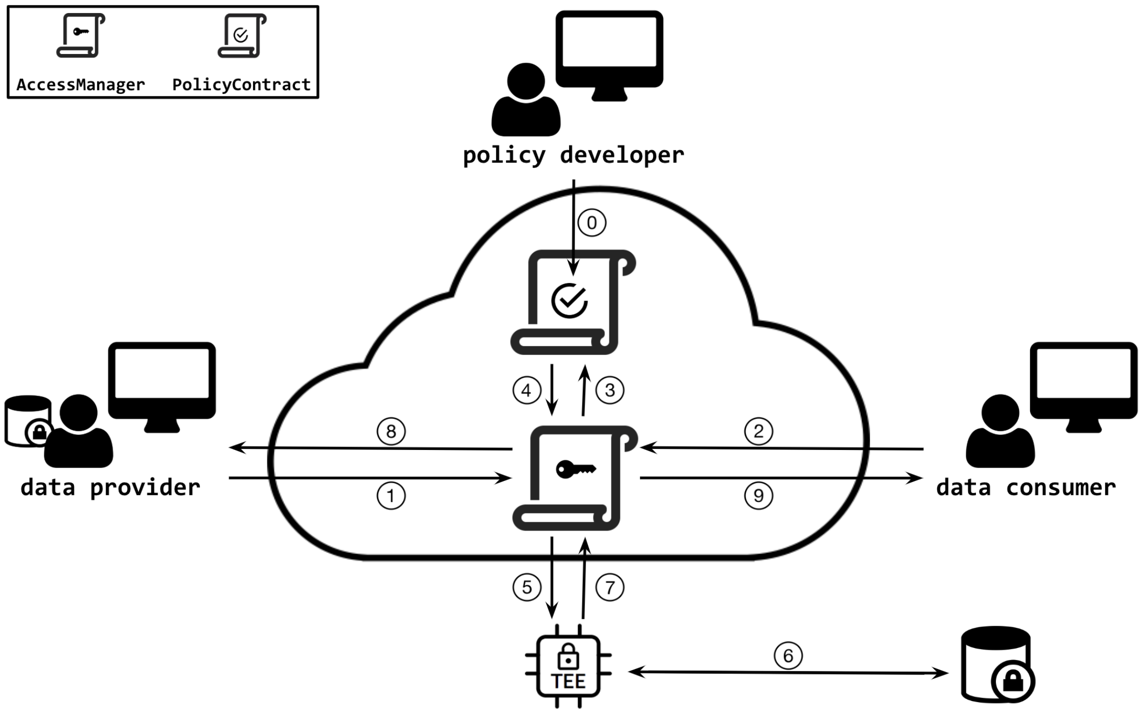


Figure 3.1.: Simple contract ecosystem

The basic interaction with the decentralized Sentinel platform is shown above. There are two primary components:

1. `Policy*` contracts, uploaded by the policy developer
2. `AccessManager`, the interaction hub between data consumers and providers.

A standard interaction, as shown in 3.1, might happen as follows:

0. Policy developer deploys policy contract `Policy0`.
1. Data provider uploads key K for dataset D to the `AccessManager` along with a database reference, and stores $\text{Enc}_K(D)$ in the linked database.
2. Data consumer attempts to execute a program P on D.
3. `AccessManager` validates that the data consumer has permission to receive $P(D)$ via `Policy0`.

4. `Policy0` affirms or denies access permissions. In this example flow, we assume an affirmation.

5. `AccessManager` provisions K and the appropriate database reference to a TEE T after correct attestation.

6. T fetches $\text{Enc}_K(D)$ from the linked database, decrypts it with K, and computes $P(D)$.

7. T returns $P(D)$ to `AccessManager`.

8. `AccessManager` notifies the data provider of the attempted access, and the corresponding result.

9. `AccessManager` returns the computational result to the data consumer.

This design was selected with flexible and robust updates in mind. Data providers can push updates to the policies they want to enforce and policy developers can push version updates to their policy contracts *independently*. This is particularly compelling for use cases such as HIPAA compliance for medical data sharing. HIPAA policies are constantly subject to review and revision, and under this model developers can easily update accordingly so long as they continue to support the API defined in 3.5.

3.3. Components

The basic solution presented in the previous section, while serving as a useful abstraction for understanding platform interaction, does not scale well to support more than a handful of users. Since platform usability is a major goal of this project, we now present a more complete and scalable interaction flow in Figure 3.2. The various interacting components of SENTINEL and their delegated responsibilities are described below:

Shared Database Before a provider and consumer can interact, they must agree upon a shared offchain database to which encrypted data can be pushed and shared. We assume that this shared database is honest-but-curious, but this is not stricitly necessary. It is a fairly straightforward extension to preserve integrity in the event of a dishonest shared database by using digital signatures, but we omit this for simplicity. By default, this offchain database is one provisioned by Oasis Labs.

Policies Policies are the security protocols providers can specify to limit access to their data. The policies can restrict access to data in a variety of different ways. Examples of such are the following:

1. Access control lists
2. Row/column filters for data tables
3. Enforcing differential privacy
4. Enforcing HIPAA compliance for medical data records

These policies are implemented as separate, mutable smart contracts, and are required to have the interface defined in 3.5. Each of the three functions `before()`, `after()`, and `validate()` each return database links to binaries that execute at the appropriate time during policy enforcement by TEEs. These contracts are linked to `AccessManagers` that enforce them via their deployment address, and are called by TEEs to retrieve the policy bytecode.

`Policy*` contracts are mutable in order to allow policy updates without having to update all `AccessManagers` with the new deployed address. Implementing them as separate smart contracts rather than directly encoding policies into `AccessManagers` is to allow greater flexibility and reuse as well.

AccessManager The `AccessManager` is a confidential smart contract that is written and deployed by the data provider. Each uploaded dataset is associated with an `AccessManager`, which serves as the gateway to that dataset. Each `AccessManager` is deployed as a confidential contract, and stores the following secret state:

1. Associated data ID
2. Associated data secret key
3. Owner's wallet address
4. Owner's public key
5. A mapping from consumer addresses to list of policy contract addresses, as well as a default "global" policy list to apply to all consumers

Upon receiving a request from a consumer, an `AccessManager` is responsible for generating a return ID for the consumer from which they can fetch computational results, emitting an event to notify offchain scheduler to begin computation, and retrieving the associated list of policy contract addresses to enforce.

DataRegistry Each `DataRegistry` is a smart contract that enables the discovery of new datasets. Users make calls to a `DataRegistry` to register new datasets in a public searchable domain and discover the datasets of others.

Compute Workers SENTINEL supports offchain compute workers, which are TEEs that run an interpreter with a very rigid execution path. These workers are managed by a scheduler, which has two responsibilities: (1) monitor the blockchain for specific events indicating that an `AccessManager` is requesting an offchain computation and (2) queue up the next available TEE to perform the job, and forward the event data to the assigned TEE. If there are no TEEs available, the scheduler stores the job and current block, and blocks on further blockchain parsing until one becomes available.

Offchain TEEs obey the following execution path:

1. Retrieve event data read by the scheduler. This includes the following, encrypted with the TEE's public key:
   - Program ID
   - Program secret key
   - Data ID
   - Data secret key
   - List of policy contract addresses
   - Return ID for the consumer
   - Consumer public key
   - Return ID for the provider
   - Provider public key

   All IDs are pointers into the shared database to encrypted data.
2. Use the event data to retrieve and decrypt the program bytecode with the program secret key and the data with the data secret key.

3. Query each of the policy contracts for pointers to their `before()`, `after()`, and `validate()` executables, and retrieve them.

4. Perform the following execution using the TEE's built-in interpreter:

    - Run each `before()` function of each policy in order of how they are listed on the decrypted data.
    - Run each `validate()` function of each policy in order of how they are listed on the consumer program.
    - Run each `after()` function of each policy in order of how they are listed on the result of applying the consumer program to the decrypted data.
    - Encrypt `success` or `failure` using the provider's public key and upload it to the shared database at the specified provider return ID.
    - Encrypt the computational result and the program ID using the consumer's public key, append its signature, and upload it to the shared database at the specified consumer return ID.
    - Notify the scheduler of completion.

Both the consumer and provider are able to audit this execution path using remote attestation quotes to verify the code and data to verify correct TEE initialization.

3.4. Scalable Contract Ecosystem

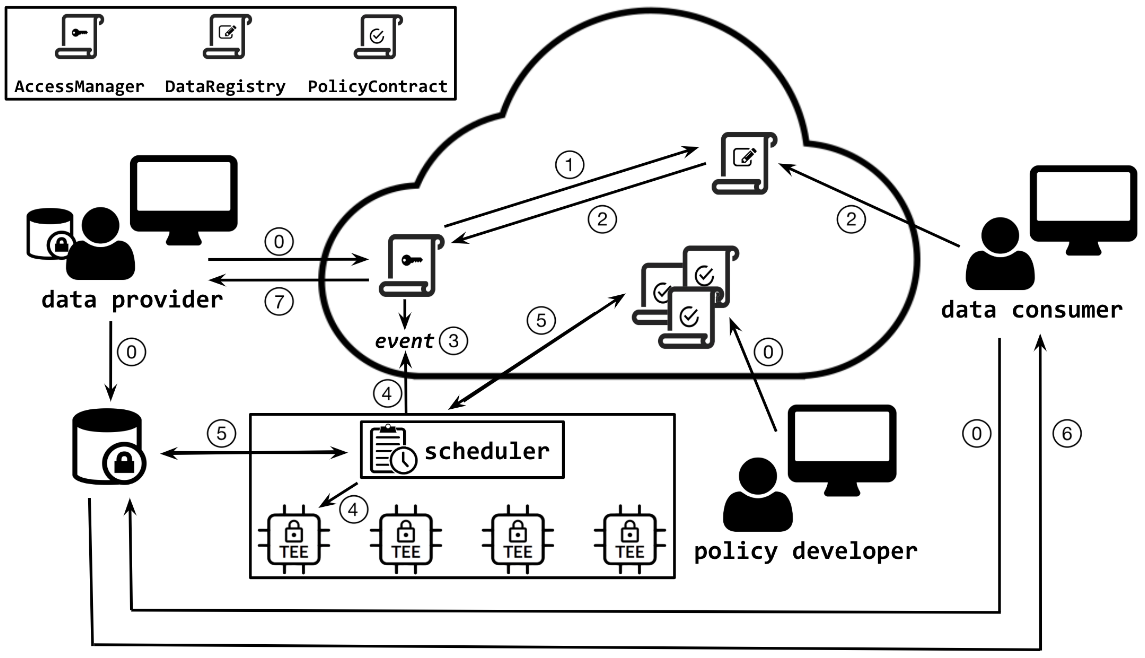


Figure 3.2.: Scalable contract ecosystem

Under this architecture, a separate, `AccessManager` contract is deployed for each uploaded dataset *independently*. Providers who wish to make their datasets searchable by consumers can register it to a global `DataRegistry` contract. Policy developers, similar to the basic model, can deploy `Policy*` contracts which can then be referenced by one or more `AccessManagers`. Collectively, these contracts form the backbone of the **contract ecosystem** for decentralized interaction between data providers and consumers, with a more comprehensive example workflow detailed below:

0. **Initialize.**
    - Policy developers write and deploy policy contracts.
    - The data provider encrypts a dataset and uploads it to the shared database.
    - The data provider writes and deploys an `AccessManager` contract, linking it with his or her dataset and policies.
    - The data consumer writes and compiles a program, then encrypts and uploads the bytecode to the shared database.

1. **Publish.** The data provider can, but is not required to, publish their dataset by registering their corresponding `AccessManager` contract with the `DataRegistry`.

2. **Request.** The data consumer sends a confidential contract call to the `AccessManager`. In its request, it includes the program ID, program secret key, and consumer public key.

3. **Process.** Upon receiving a request, the `AccessManager` emits an event for the scheduler, generates a return ID that points to the shared database location wher the computational result will be written, and returns it to the consumer. The emitted event contains everything listed in TEE section above.

4. **Schedule.** The scheduler, monitoring the blockchain, reads an event indicating that an `AccessManager` is requesting an offchain computation. It will schedule a TEE to perform the job via a round-robin policy.

5. **Compute.** Upon being queued by the scheduler, the assigned TEE performs the execution path described above, retrieving and decrypting the appropriate data, consumer code, and policies and performing the appropriate computation. The result(s) are encrypted with the consumer public key, signed by the TEE and is stored in the shared dataset at the consumer return ID generated in step 3.

6. **Retrieve.** The consumer monitors the shared database until the computational result is written, then they can decrypt it with their secret key and verify that it was written by a TEE.

7. **Audit.** The provider can query the blockchain to retrieve audit logs of accesses to their datasets.

These design choices were made with current limitations of smart contracts in mind. Due to expensive gas costs that scale infeasibly with contract state complexity, minimizing the number of computational steps required in each state update to a contract is of utmost importance. Hence each `AccessManager` is responsible only for storing a single key and a mapping from user identities to policies. This contrasts with the basic solution in Section 3.2, where a single `AccessManager` is responsible for managing keys for multiple datasets and so does not scale to support more than a handful of users and datasets.

3.5. Governance APIs

In this section, we denote the governance APIs by which data producers and consumers interact with the contract ecosystem more concretely. The API for interacting with the AccessManager is described below:

```
// API for provider interaction, can only be called by the AM creator
/// Creates a new `AccessManager` with a pointer to data, and a key.
fn new(&mut self, data_id: String, data_key: String, pub_key: String) ->
    Result<Self>;

/// Suicides the given AccessManager
fn delete(&mut self) -> bool;

/// Registers/unregisters the provided policy to each of the provided users.
/// If the list of users is empty, the policy is applied globally.
fn register_policy(&mut self, policy: Address, user: Address) -> bool;
fn unregister_policy(&mut self, policy: Address, users: Address) -> bool;

// API for consumer interaction
/// Attempts to run a program on the data managed by this AM.
/// Returns a handle to a resource that contains the results on completion.
/// This handle is a null value if the sender() is not whitelisted for this
    dataset.
fn execute(&mut self, program_id: String) -> Result<String>;
```

Each dataset has an AccessManager which can only be deleted by the data owner, and only the owner can add or remove policies to enforce for any other user. AccessManagers also have an associated trusted enclave with which they attest before offloading decryption keys, executables, policies, etc. Finally, each AccessManager supports cross-contract calls to request provisioning of policies enforced for any whitelisted consumers to the appropriate TEE before performing any given consumers' requested computation. These policy contracts support the following API:

```
/// Creates a new `Policy` with a list of users to apply to.
/// The parameters are pointers to Wasm binaries stored in a database.
fn new(&mut self, before_id: String, after_id: String, validate_id: String)
    -> Result<Self>;

// API for policy developer interaction
/// Update the stored policy. Can only be done by policy developer.
fn update_validate_bytecode(&mut self, new_program_id: String) -> bool;
fn update_before_bytecode(&mut self, new_program_id: String) -> bool;
fn update_after_bytecode(&mut self, new_program_id: String) -> bool;

// API for TEE interaction.
/// Calls to any of these functions returns a handle to the shared database
/// location containing Wasm binary for the corresponding function.
fn validate(&mut self) -> Result<String>;
fn before(&mut self) -> Result<String>;
fn after(&mut self) -> Result<String>;
```

Once an access policy is provisioned to a trusted compute worker and the dataset is loaded and decrypted, the policy is applied via the functions implemented in the policy API, which as described must be implemented by the policy developer in any `Policy*` contract.

1. `before()` - A function applied before program execution. A policy acts on dataset D to produce $D' = \texttt{before}(D)$. For example, P might be a machine learning algorithm and `before()` might be a filter that removes attributes from a dataset prior to training.

2. `after()` - A function applied after program execution. A policy acts on $P(D')$ to produce $\texttt{after}(P(D'))$ which is then received by the consumer. For example, P might be a SQL query and `after()` might be a function that adds random noise to the query result to make it differentially private.

3. `program()` - A function applied to programs themselves. A policy computes $\texttt{program}(P)$, which returns true if the program is allowable to execute over D and false otherwise. For example, P might be a SQL query that uses the `GROUP BY` keyword, but the referenced policy blacklists SQL queries containing `GROUP BY`.

In summary, the TEE's responsibility consists of two distinct steps: (1) validating `assert(program(P))` is true and (2) computing `after(P(before(D)))` and returning the result to the calling consumer. The generality of the governance API is intentional; with such loose specifications policy contracts implemented by developers will be exceedingly use-case driven and insightful patterns will arise organically.

4. Privacy

In this section, we motivate the medical data collaboration use case by providing an analysis for building a privacy-preserving `Policy*` contract. We discuss relevant and typical privacy-preserving database queries for policies involving differential privacy for n mutually untrusting medical data providers who wish to collaborate for their complementary benefit using SENTINEL. More formally, we consider providers with datasets $x_1, \ldots, x_n \in \mathbb{N}^{|\mathcal{X}|}$ and corresponding privacy budgets $\epsilon_1, \ldots, \epsilon_n$, and analyze typical classes of medical queries over an aggregate database $x = \bigcup_{i=1}^{n} x_i$ which satisfy each of the individual privacy budgets.

4.1. Histogram Queries

Histogram queries are counting queries that retrieve the counts in each cell after a partitioning of the database. In particular, we consider the problem of finding the *most common medical condition* amongst patients aggregated from multiple databases.

Standard histogram queries can be made $(\epsilon, 0)$-differentially private via the **Laplace mechanism**.

Definition 4.1.1 (The Laplace Mechanism)**.** Given any function $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$ the Laplace mechanism is defined as

$$\mathcal{M}_{LAP}(x, f(\cdot), \epsilon) = f(x) + (Y_1, \ldots, Y_k)$$

where each Y_i are i.i.d. variables drawn from the Laplace distribution, $Y_i \sim \mathrm{Lap}(\Delta f/\epsilon)$.

The Laplace distribution is known to preserve $(\epsilon, 0)$-differential privacy as desired, a fact which is shown in A.1.

Since addition of a new user can change the count in exactly one histogram cell by exactly 1 for any given query, applying the Laplace mechanism with l_1-sensitivity 1 ensures that the query is $(\epsilon, 0)$-differentially private. However, solving the most common medical condition problem for m medical conditions while enforcing $(\epsilon, 0)$-differential privacy has the potential to be highly sensitive and require a high privacy budget, as patients can have several medical conditions and so removal of a single one can change the counts in *several* of the m needed histogram queries. We now show that with our n database construction, selecting $\epsilon = \max(\epsilon_1, \ldots, \epsilon_n)$ we can achieve $(\epsilon, 0)$-differential privacy by applying the Laplace mechanism with the very reasonable sensitivity $\Delta f = 1$ to each query. Referencing [3] heavily, we can make and prove the following claim:

Claim. *Adding noise sampled from Lap$(1/\epsilon)$ to each count and returning the index of the largest count is $(\epsilon, 0)$-differentially private in aggregate.*

Proof. Let D' be a database, and $D = D' \cup \{a\}$. Let c and c' denote the vectors of histogram counts when the databases under consideration are D and D', respectively. We keep the following two properties in mind:

- *Monotonicity of Counts.* For all $j \in [m]$, $c_j \geq c'_j$.
- *Lipschitz Property.* For all $j \in [m]$, $1 + c'_j \geq c_j$.

Fixing any $i \in [m]$ we now bound the ratio of the probabilities that i is selected with D and with D' both from above and below. Let r_{-i} represent a draw from $[\text{Lap}(1/\epsilon)]^{m-1}$, which is used to add noise to all counts but the i^{th} histogram cell. Defining $\Pr[i|\xi]$ as the probability that the condition at index i is selected as the most common condition, conditioned on x_i, we now argue the privacy properties of each r_{-i} independently.

Let's first argue that $\Pr[i|D, r_{-i}] \leq e^{\epsilon}\Pr[i|D', r_{-i}]$. Define

$$r^* = \min_{r_i} \ : \ c_i + r_i > c_j + r_j \ \ \forall j \neq i$$

Since we've fixed r_{-i}, i will be output as the most common medical condition in database D if and only if $r_i \geq r^*$. We have that $\forall\ 1 \leq j \neq i \leq m$:

$$\begin{aligned} c_i + r^* &> c_j + r_j \\ \implies (1 + c'_i) + r^* \geq c_i + r^* &> c_j + r_j \geq c'_j + r_j \\ \implies c'_i + (r^* + 1) &> c'_j + r_j \end{aligned}$$

Hence, if $r_i \geq r^* + 1$, then the i^{th} count will be maximized with database D' and noise vector (r_i, r_{-i}). We can now complete the proof of the forward direction as follows, where each $r_i \sim \text{Lap}(1/\epsilon)$ below.

$$\begin{aligned} \Pr[r_i \geq 1 + r^*] &\geq e^{-\epsilon}\Pr[r_i \geq r^*] = e^{-\epsilon}\Pr[i|D, r_{-i}] \\ \implies \Pr[i|D', r_{-i}] \geq \Pr[r_i \geq 1 + r^*] &\geq e^{-\epsilon}\Pr[r_i \geq r^*] = e^{-\epsilon}\Pr[i|D, r_{-i}] \end{aligned}$$

We conclude that $\Pr[i|D, r_{-i}] \leq e^{\epsilon}\Pr[i|D', r_{-i}]$ as desired.

The reverse direction can be shown in a similar manner, and so we argue that $\Pr[i|D', r_{-i}] \leq e^{\epsilon}\Pr[i|D, r_{-i}]$, defining

$$r^* = \min_{r_i} \; : \; c'_i + r_i > c'_j + r_j \; \forall j \neq i$$

Again, now that we have fixed r_{-i}, i will be output as the most common medical condition in database D' if and only if $r_i \geq r^*$. We have that $\forall\ 1 \leq j \neq i \leq m$:

$$\begin{aligned} c'_i + r^* &> c'_j + r_j \\ \implies 1 + c'_i + r^* &> 1 + c'_j + r_j \\ \implies c'_i + (r^* + 1) &> (1 + c'_j) + r_j \\ \implies c_i + (r^* + 1) \geq c'_i + (r^* + 1) &> (1 + c'_j) + r_j \geq c_j + r_j \end{aligned}$$

which allows us to conclude that when $r_i \geq r^* + 1$ the maximum count medical condition will be selected as the one at index i when we have database D with noise vector (r_i, r_{-i}). Hence we see that

$$\Pr[i|D, r_{-i}] \geq \Pr[r_i \geq r^* + 1] \geq e^{-\epsilon}\Pr[r_i \geq r^*] = e^{-\epsilon}\Pr[i|D', r_{-i}]$$

We conclude that $\Pr[i|D', r_{-i}] \leq e^{\epsilon}\Pr[i|D, r_{-i}]$ as desired, and since we selected ϵ as the maximum of all individual privacy constraints, each individual database's privacy budget is respected. □

4.2. Utility Queries

Utility queries are queries where, unlike counting queries, a utility function is applied to outputs and a small amount of noise over the resultant computed quantity can significantly change its utility. Notably, these queries consist of situations where the output space consists of an arbitrary range rather than $\mathbb{N}$ as with counting queries. Specifically, we consider the problem of *optimal dosage pricing*, the best price at which to sell a pharmaceutical drug to maximize revenue.

Standard utility queries can be made $(\epsilon, 0)$-differentially private via the **exponential mechanism**.

Definition 4.2.1 (The Exponential Mechanism)**.** Given any arbitrary range $\mathcal{R}$ and utility function $u : \mathbb{N}^{|\mathcal{X}|} \times \mathcal{R} \rightarrow \mathbb{R}^k$ the exponential mechanism is defined as

$$\mathcal{M}_{EXP}(x, u(\cdot, \cdot), \mathcal{R})$$

where each $r \in \mathcal{R}$ is selected with probability proportional to $\exp(\frac{\epsilon u(x,r)}{2\Delta u})$. Δu represents the sensitivity of utility function u, defined as follows:

$$\Delta u = \max_{r \in \mathcal{R}} \max_{x, y : ||x-y||_1 \leq 1} |u(x, r) - u(y, r)|$$

We consider now that each of our databases x_i consists of tuples (d_{ij}, p_{ij}) the required dosage for each user and the maximum price they are willing to pay per unit (i.e. milligram) for a given drug. Naturally, the higher the price, the fewer users will buy it, but the lower the price, the lower the profit margins; thus, optimal pricing is important. Selecting $\epsilon = \min(\epsilon_1, \ldots, \epsilon_n)$ we prove that we can determine a "good" dosage price with high probability, where good is determined in terms of the highest-utility price:

$$\text{OPT}_u(x) = \max_{r \in \mathcal{R}} u(x, r)$$

Under the exponential mechanism, the utility of the output price can differ from $\text{OPT}_u(x)$ by no more than an additive factor of $O((\Delta u)/\epsilon) \log |\mathcal{R}|)$. Referencing [3] heavily, we make the following claim:

Claim. *Fixing a database x and let $\epsilon = \min(\epsilon_1, \ldots, \epsilon_n)$. Then:*

$$Pr\left[u(\mathcal{M}_E(x, u, \mathcal{R})) \leq OPT_u(x) - \frac{2\Delta u}{\epsilon}\left(\ln |\mathcal{R}| + t\right)\right] \leq e^{-t}$$

Proof. Let $\mathcal{R}_{\text{OPT}} = \{r \in \mathcal{R} \,:\, u(x,r) = \text{OPT}_u(x)\}$, the set of elements of $\mathcal{R}$ which attain the optimal utility score $\text{OPT}_u(x)$. Then, we can observe that:

$$\begin{aligned}
\Pr[u(\mathcal{M}_{EXP}(u,x,\mathcal{R}) \le c)] &\le \frac{|\mathcal{R}| \exp(\epsilon c/2\Delta u)}{|\mathcal{R}_{\text{OPT}}| \exp(\epsilon \text{OPT}_u(x)/2\Delta u)} \\
&= \frac{|\mathcal{R}|}{|\mathcal{R}_{\text{OPT}}|} \exp\left(\frac{\epsilon(c - \text{OPT}_u(x))}{2\Delta u}\right) \\
&\le |\mathcal{R}| \exp\left(\frac{\epsilon(c - \text{OPT}_u(x))}{2\Delta u}\right)
\end{aligned}$$

The first step above follows from noting that each $r \in \mathcal{R}$ has, by definition, unnormalized probability mass bounded by $\exp(\epsilon c/2\Delta u)$ when $u(x,r) \le c$. This bounds the total probability of elements with utility not equal to $\text{OPT}_u(x)$ from above by $|\mathcal{R}| \exp(\epsilon c/2\Delta u)$. The final step comes from observing that $|\mathcal{R}_{\text{OPT}}| \ge 1$ and so $\exp(\epsilon \text{OPT}_u(x)/2\Delta u)$.

This allows us to show the desired bound:

$$\Pr\left[u(\mathcal{M}_{EXP}(x,u,\mathcal{R})) \le \text{OPT}_u(x) - \frac{2\Delta u}{\epsilon}(\ln|\mathcal{R}| + t)\right] \le e^{-t}$$

with appropriate selection of c. Using our definition of ϵ, we have:

$$\text{OPT}_u(x) - \frac{2\Delta u}{\epsilon}(\ln|\mathcal{R}| + t) \ge \text{OPT}_u(x) - \frac{2\Delta u}{\epsilon_i}(\ln|\mathcal{R}| + t) \ \ \forall i \in [n]$$

and so

$$\begin{aligned}
\Pr\left[u(\mathcal{M}_{EXP}E(x,u,\mathcal{R})) \le \text{OPT}_u(x) - \frac{2\Delta u}{\epsilon_i}(\ln(|\mathcal{R}|) + t)\right] &\le \Pr\left[u(\mathcal{M}_{EXP}(x,u,\mathcal{R})) \le \text{OPT}_u(x) - \frac{2\Delta u}{\epsilon}(\ln(|\mathcal{R}|) + t)\right] \\
&\le e^{-t}
\end{aligned}$$

Hence, the desired privacy budget requirements for each of the n databases is met if we set $\epsilon = \min \epsilon_i \ \ \forall i \in [n]$. □

4.3. Composition

In the previous two sections, we have discussed use cases for aggregating datasets while respecting privacy budgets of each dataset independently. It is also important to consider a second means for aggregation: *composition.* Composition refers to making several independent queries to a single dataset in sequence, and a standard goal is to prevent the aggregate information gained from the union of all queries to violate the desired privacy constraints. This is handled gracefully by the rule "the ϵs and δs add" [3] - the net privacy budget used by a sequence of queries is simply the sum of the individual ϵs and δs of the queries. Hence a `Policy*` contract enforcing (ϵ, δ)-differential privacy can simply track this net privacy budget and simply refuse to answer any additional queries once it has been exceeded. In fact, chaining queries together can be optimized further to be frugal with expending privacy budget via the *strong composition theorem*:

Theorem 1 (Strong Composition Theorem)**.** *For every ϵ, δ, $\delta' > 0$, and $k \in \mathbb{N}$, the class of (ϵ, δ)-differentially private mechanisms is $(\epsilon', k\delta + \delta')$-differentially private under k-fold adaptive composition, for*

$$\epsilon' = \sqrt{2k \ln(1/\delta')} \cdot \epsilon + k \cdot \epsilon\epsilon_0$$

where $\epsilon_0 = e^{\epsilon} - 1$

which is derived and proven in [4].

5. Implementation & Usability

This section provides relevant implementation details and performance optimizations required to construct the proposed system architecture in a usable manner.

5.1. WebAssembly and Contract Kit

WebAssembly (abbreviated Wasm) is a binary instruction format which is a target for compiling high-level languages, which for our implementation refers to Rust. Wasm is practical because it tries to match CPU semantics rather than language semantics, and so languages that compile to Wasm tend to avoid the lack of tooling and support that plagues many Ethereum Virtual Machine (EVM) compatible languages such as Solidity.

Wasm makes it possible to develop **confidential contracts**, smart contracts with private state, in Rust within Oasis Labs' `contract-kit` framework, a Docker image that makes it easy to test and deploy these contracts. When compared to Solidity, using Rust enables policy developers almost unbounded flexibility for the richness of policies they can implement and make available to data providers.

For conducting blockchain transactions, we utilize `web3c`, Oasis Labs' wrapper around `web3` (the Ethereum Javascript API). While standard blockchain transactions are transparently visible publicly, `web3c` transactions are end-to-end encrypted allowing secure communication with contracts via the underlying RPC calls.

5.2. Oasis Developer Dashboard

Each action taken through Sentinel requires making one or more blockchain transactions with the contract ecosystem, and each of these transactions requires Oasis DEV tokens to complete. The Oasis Developer Dashboard is a developer tool used to facilitate smart contract management and deployment. It is used to deploy contracts within our data governance contract ecosystem, and provides two critical functions that are important for usability:

- *User Activity Analytics* - The Developer Dashboard registers users to contracts with which they have interacted, and tracks how many tokens have been interacting with the contract ecosystem.
- *Autofunding* - The Developer Dashboard periodically checks the number of tokens possessed by each active user's account and funds it up to 1 DEV each day if they are significantly below this limit.

Together, these two functions ensure that users always have sufficient funds required for interaction with the SENTINEL platform for a seamless user experience.

5.3. Soft Wallets

The current standard for executing transactions on the Ethereum blockchain is via Metamask, a secure identity vault which provides a user interface for both managing your identities across different websites and signing transactions for you. However, executing transactions in this manner makes building SENTINEL infeasible, as it requires an unrealistic number of clicks to function as a usable web application.

Instead, SENTINEL utilizes **soft wallets** - wallets generated for users upon initial login to the platform, stored and managed entirely client-side in users' browsers, and automatically funded periodically by Oasis Labs. Any interactions with the platform taken by a user are authorized via transactions initiated via their soft wallets under-the-hood and entirely abstracted away, so that the application looks and feels like a traditional data sharing application (i.e. Google Drive).

Soft wallets are created via `web3c` by generating new Ethereum accounts, which are then registered to relevant contracts within the contract ecosystem via the Developer Dashboard, and after which they will automatically and regularly receiving funding in the form of Oasis DEV tokens to interact with SENTINEL. The generated account's private key is stored and persisted via the user's browser `localStorage`, a persistent key-value store unique for each webpage origin, and is used to unlock the corresponding soft wallet each time the user logs in.

6. Evaluation

In this section, we validate the performance and usability of Sentinel.

6.1. Experimental Setup

We evaluate Sentinel in two distinct ways: measuring end-to-end latency and costs for specific operations from our governance API, and by evaluating the utility and quality of de-siloed data constrained by access policies through Sentinel for running a simple machine learning pipeline. End-to-end latency is measured for a user using a standard 2.9 GHz Intel Core i7 processor to interact with our platform. For the latter, we train a simple multi-layered perceptron (MLP) both using siloed datasets and also with Sentinel using aggregated datasets with specified access policies. These metrics are computed using a proof-of-concept that implements the basic solution presented in 3.2, and validating that the platform is still satisfactorily usable. This bodes well, as usability can only increase with the scalable contract ecosystem and other optimizations.

6.2. End-to-end Latency

We measure and tabulate gas costs for basic data governance operations. These costs are fixed and deterministic given operation parameters and internal contract state, since they are a function of the number of computational steps taken.

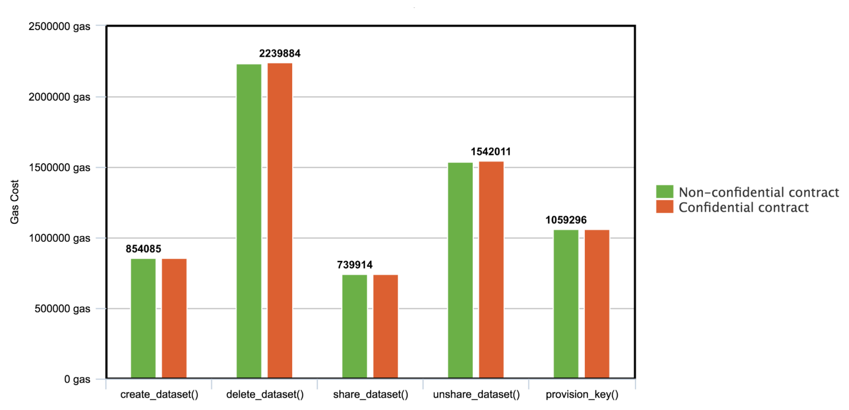


Figure 6.1.: Gas costs for basic governance operations

Notably, the gas costs are quite close in both the confidential and non-confidential cases, with the difference resulting from overhead associated with encryption of confidential state. Additionally, we measure the end-to-end latency of performing these operations, both in the confidential and non-confidential cases.

Table 6.1.: End-to-End Operation Latency

Operation	**Non-confidential Contract**	**Confidential Contract**
Create project	2647ms	5200ms
Delete project	2594ms	5135ms
Create dataset	2562ms	4531ms
Delete dataset	3498ms	6218ms
Add policy	2594ms	5305ms
Remove policy	2994ms	6475ms
Successful data access	3495ms	5541ms
Failed data access	4161ms	6638ms
Fetch audit log	1525ms	9111ms

Overall, transactions on the Ethereum blockchain take 15 seconds on average, and so these margins all significantly outperform those metrics. Indeed, even in the considerably slower confidential case the time for operations is better than this by a factor of nearly 3. All of these presented latencies are reasonable for an end user and with sufficient usage of “optimistic updates” and other UI tricks we are able to construct a highly-usable governance application backed by the Oasis blockchain.

6.3. Performance

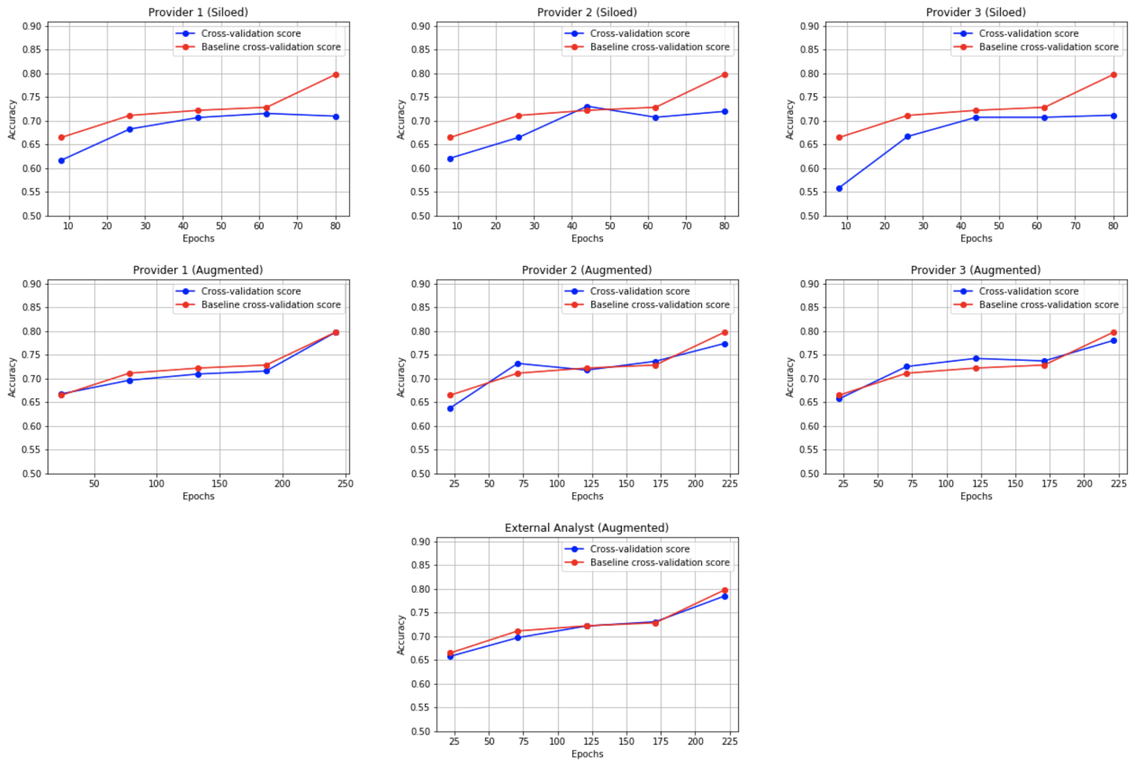


Figure 6.2.: Training performance on UCI Heart Disease dataset

To validate the utility of de-siloed data under various regulations and access policies, we train a simple multi-layer perceptron for binary classification under a spectrum of constraints via the UCI Heart Disease dataset, which contains information such as the age, sex, cholesterol, heart rate, etc. for a set of patients. This dataset is divided into 3 randomly selected partitions and assigned to each of 3 data providers, who have the following (typical) access policies for the patients whose data they are in charge of managing:

1. *Provider 1*: Provider 1 has decided that 50% of their patients' data cannot be made available to the other data providers.

2. *Provider 2*: Provider 2 has decided not to make their patients below the age of 50 and with a cholesterol level above 300 mg/dL visible to the other providers.

3. *Provider 3*: Provider 3 has opted to hide their patients' age, sex, and maximum heart rate.

We consider scenarios where each provider only trains using only their siloed data, as well as in aggregation with policy-constrained datasets via SENTINEL. Additionally, we consider the case of an external analyst, who has access only to policy-constrained datasets. Any missing values in the aggregate dataset for each case are specified using a simple mean imputer, without any special optimizations. The performance evaluation via learning curves of cross-validation accuracy for each setup can be seen in Figure 6.2, each of which are compared to the baseline learning curve derived from training over the entire UCI Heart Disease dataset without restriction. These plots are generated using results aggregated over 100 iterations of cross-validation, with 20% of data selected at random used as a validation set.

Notably, this validates conservation of data quality in typical platform uses cases. Even with considerably restrictive and robust access policies, training via SENTINEL yields performance that mirrors the baseline while training via siloed data alone yields significantly lower cross-validation scores.

7. Related Work and Future Direction

There are several future directions for further development of Sentinel, both in terms of performance optimization and policy aggregation.

This work draws upon and extends ideas initially described in [12, 18]. [12] describes a data marketplace wherein provider/consumer interactions occur via a framework similar to the proposed contract ecosystem, but where payment for data usage is the primary enforced policy. It describes data economics as a vehicle for an organic, empirical evaluation of "good quality" data. It would be an interesting case study to investigate similar use-case driven policy formalism and community standards.

Prior work in [6] proposes an efficient system for verifiable *time-window queries* and *subscription queries* rooted in extending blockchain block headers to support an authenticated data structure for verification. Using skip-lists for building an inter-block index and Jaccard similarity for building an intra-block index, the authors of [6] achieve a performant blockchain for the types of queries described. These ideas are very applicable to improve performance of the audit logging by data providers and/or supporting a notification system via subscription queries on Sentinel.

Another future avenue for work is with policy commutativity and hierarchy. When multiple policies are applied to a consumer accessing a dataset, in which order should the respective `before()` (or `after()`, or `validate()`) functions of the two policies be applied? Similarly, if the computational result of a data access is itself stored as a new dataset, is it necessary to enforce any "residual" policies over the outputs before allowing additional parties to perform any further computation? These are largely still open questions. [11] has made progress towards addressing concerns about hierarchical policies for the case of differential privacy, defining a formal type system for automatically evaluating the differential privacy of composable programs such as machine learning algorithms.

Finally, it would be interesting to evaluate TEEs in GPUs for applications such as machine learning rather than TEEs in CPUs, which are the current industry standard. [21] proposes an architecture for implementing exactly this in off-the-shelf NVIDIA GPUs, and [13] describes a similar idea using FPGA accelerators.

8. Conclusion

In this paper, we have proposed and studied Sentinel, a novel architecture for fully-decentralized data governance. This architecture achieves decentralization via a robust contract ecosystem on the Oasis blockchain and achieves high-throughput performance and usability via trusted execution environments. We have motivated this design with applications in healthcare and analyzed relevant example policies such as those that enforce differential privacy while querying over data in aggregate. Medical data sharing has traditionally been siloed and severely limited by factors such as data privacy and HIPAA compliance, even though collaboration unequivocally results in more accurate machine learning algorithms and more insightful data analytics.

This project is only the beginning, and we hope that it serves as a case study for more widespread adoption of such a platform. The social utility it has the potential to unlock is unbounded.

There are many more directions for further exploration that are promising. Deeper investigation into formalism for policy definition, in particular for policy aggregation and commutativity analysis such as in [7, 11] are areas of study that will increase the richness and robustness of Sentinel's computational support. Better optimization for state storage and access logging similar to [6] is another direction for exploration, as is increasing TEE support from CPU to GPU-based, such as in [13, 21], for optimizing performance for machine learning and other computationally intense applications.

Interested parties should reach out and email `ennsharma@berkeley.edu` to collaborate or request a demo.

9. Acknowledgements

This research project was made possible by the constant support by my colleagues at Oasis Labs, Inc. and at UC Berkeley. I want to thank Prof. Joe Near for his guidance leading me into the world of security and privacy research. I would also like to thank Raymond Cheng for his mentorship towards pursuing my research goals within the product-oriented framework at Oasis Labs.

I would like to thank all of my collaborators at Oasis Labs with whom I've worked closely towards completing this project: Clement Fung, Peter Gilbert, Will Scott, Nick Hynes, Reuven Gonzales, David Yan, and Naheed Vora.

I would also like to thank Victor Chan and Alex Wang for their contributions to this project via the Spring 2019 offering of Graduate Algorithms (CS 270) at UC Berkeley.

Finally, thank you to Prof. Dawn Song, for serving as my research advisor during the duration of my Master's degree. Without her enthusiastic support and interest in hearing my ideas and fostering my personal growth both as an undergraduate and graduate student, none of this would be possible.

Thank you to all these people, whose individual excellence inspires me to grow and be better every day as a researcher, engineer, and educator.

Bibliography

[1] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, *Hawk: The blockchain model of cryptography and privacy-preserving smart contracts*, in IEEE Security and Privacy, 2016.

[2] A. Miller, M. Hicks, J. Katz, E. Shi, *Authenticated Data Structures, Generically*, POPL, 2014.

[3] C. Dwork, A. Roth, *The Algorithmic Foundations of Differential Privacy, Foundations and Trends in Theoretical Computer Science*, v.9 n.3-4, p.211-407, August 2014. DOI: https://doi.org/10.1561/0400000042

[4] Cynthia Dwork , Guy N. Rothblum , Salil Vadhan, Boosting and Differential Privacy, Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, p.51-60, October 23-26, 2010. https://doi.org/10.1109/FOCS.2010.12

[5] C. Papamanthou, R. Tamassia, and N. Triandopoulos. *Optimal verification of operations on dynamic sets.* In Crypto, pages 91–110, 2011.

[6] C. Xu, C. Zhang, and J. Xu, *vChain: Enabling Verifiable Boolean Range Queries Over Blockchain Databases*, in ACM SIGMOD, Amsterdam, Netherlands, 2019.

[7] F. Wang, R. Ko, and J. Mickens, *Riverbed: Enforcing User-defined Privacy Constraints in Distributed Web Services.* In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19) (2019), USENIX, p. To appear.

[8] G. Zyskind, O. Nathan et al., *Decentralizing privacy: Using blockchain to protect personal data*, in Security and Privacy Workshops (SPW), 2015 IEEE. IEEE, 2015, pp. 180–184.

[9] H. Ebadi, D. Sands, G. Schneider, *Differential Privacy: Now it's Getting Personal*, Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, January 15-17, 2015, Mumbai, India. DOI: https://doi.org/10.1145/2676726.2677005

[10] H. Corrigan-Gibbs and D. Boneh. 2017. *Prio: Private, Robust, and Scalable Computation of Aggregate Statistics.* In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). USENIX Association, Boston, MA, 259–282. URL: `https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs`

[11] J. P. Near, D. Darais, T. Stevens, et al. *Duet: A Language and Type System for Statically Enforcing (ϵ, δ)-Differential Privacy.* 2019.

[12] N. Hynes, D. Dao, D. Yan, R. Cheng, D. Song. A Demonstration of Sterling: A Privacy-Preserving Data Marketplace. *PVLDB* 11 (12): 2086-2089, 2018. DOI: `https://doi.org/10.14778/3229863.3236266`

[13] N. Hynes et al. *A Tensor Encryption Core Suitable for Machine Learning Accelerators.* 2019.

[14] N. Johnson, J. P. Near, and D. Song. *Towards Practical Differential Privacy for SQL Queries.* PVLDB, 11(5):526–539, 2018.

[15] N. Johnson, J. P. Near, J. Hellerstein, D. Song. (2018). *Chorus: Differential Privacy via Query Rewriting.*

[16] *Oasis Blockchain.* 2018. *Oasis Labs, Inc.*. URL: `https://www.oasislabs.com/`.

[17] R. Canetti, O. Paneth, D. Papadopoulos, and N. Triandopoulos. *Verifiable set operations over outsourced databases.* In PKC, pages 113–130, 2014.

[18] R. Cheng et al. *Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contract Execution.* In: *arXiv:1804.05141* (2018).

[19] R. Cummings, S. Krehbiel, K. A. Lai, and U. Tantipongpipat. *Differential Privacy for Growing Databases.* CoRR, abs/1803.06416, 2018.

[20] R. Munz, F. Eigner, M. Maffei, P. Francis, D. Garg (2018). *UniTraX: Protecting Data Privacy with Discoverable Biases.* In: Bauer L., Küsters R. (eds) *Principles of Security and Trust.* POST 2018. Lecture Notes in Computer Science, vol 10804. Springer, Cham.

[21] S. Volos, K. Vaswani, and R. Bruno. 2018. *Graviton: Trusted Execution Environments on GPUs.* In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18). 681–696.

[22] V. Buterin. *Ethereum White Paper.* 2013. *Ethereum.*

[23] Y. Zhang, J. Katz, and C. Papamanthou. IntegriDB: Verifiable SQL for outsourced databases. In Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS), Denver, CO, 2015.

A. Theorems & Lemmas

Here we show that applying the Laplace and exponential mechanisms to certain classes of functions preserves $(\epsilon, 0)$-differential privacy, originally from [3].

A.1. The Laplace Mechanism

Theorem 2. *The Laplace mechanism preserves* $(\epsilon, 0)$*-differential privacy.*

Proof. Let $x, y \in \mathbb{N}^{|\mathcal{X}|}$ be such that $||x - y||_1 \leq 1$, and define $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$. Let p_x and p_y denote the probability density functions of $\mathcal{M}_{LAP}(x, f, \epsilon)$ and $\mathcal{M}_{LAP}(y, f, \epsilon)$, respectively. Comparing the two at an arbitrary point $z \in \mathbb{R}^k$, we can derive

$$\begin{aligned}
\frac{p_x(z)}{p_y(z)} &= \prod_{i=1}^{k} \left(\frac{\exp(-\frac{\epsilon |f(x)_i - z_i|}{\Delta f})}{\exp(-\frac{\epsilon |f(y)_i - z_i|}{\Delta f})} \right) \\
&= \prod_{i=1}^{k} \exp\left(\frac{\epsilon(|f(y)_i - z_i| - |f(x)_i - z_i|)}{\Delta f} \right) \\
&\leq \prod_{i=1}^{k} \exp\left(\frac{\epsilon |f(x)_i - f(y)_i|}{\Delta f} \right) \\
&= \exp\left(\frac{\epsilon \cdot ||f(x) - f(y)||_1}{\Delta f} \right) \\
&\leq \exp(\epsilon)
\end{aligned}$$

Above, the first inequality is simply an application of the triangle inequality, and the last inequality stems directly from the definition of sensitivity and our given that x and y are neighboring databases. By symmetry, it follows that $\frac{p_y(z)}{p_x(z)} \geq \exp(-\epsilon)$. □

A.2. The Exponential Mechanism

Theorem 3. *The exponential mechanism preserves* $(\epsilon, 0)$*-differential privacy.*

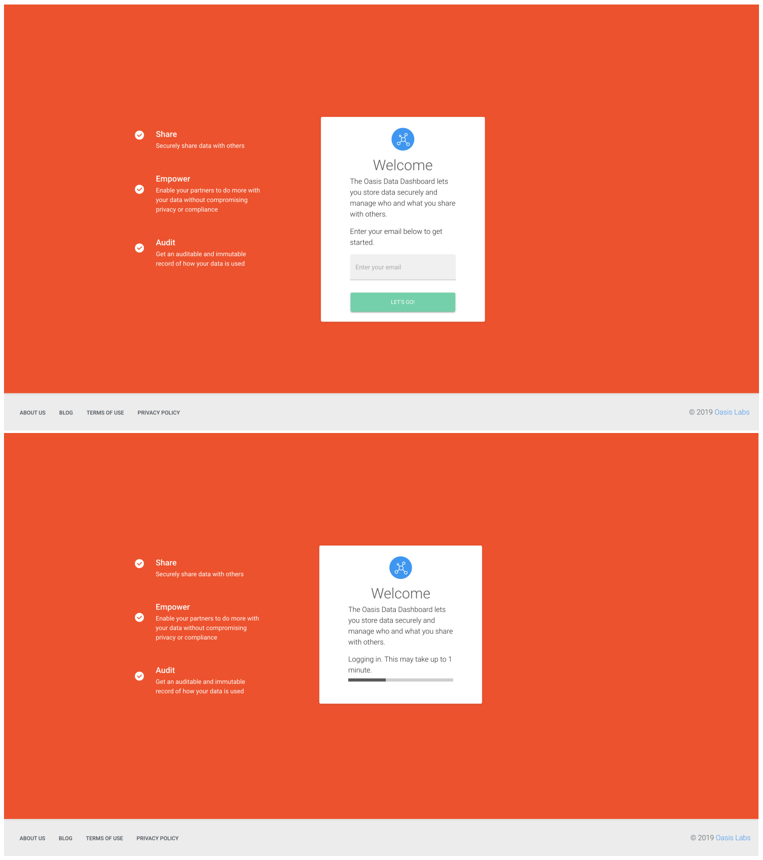
Proof. Let $x, y \in \mathbb{N}^{|\mathcal{X}|}$ be such that $||x - y||_1 \leq 1$. Define a range $\mathcal{R}$ representing the output space of the exponential mechanism and a given utility function which maps from database/output pairs to utility scores $u : \mathbb{N}^{|\mathcal{X}|} \times \mathcal{R} \to \mathbb{R}$. Let p_x and p_y denote the probability density functions of $\mathcal{M}_{EXP}(x, u, \mathcal{R})$ and $\mathcal{M}_{EXP}(y, u, \mathcal{R})$, respectively. Comparing the two at an arbitrary point $r \in \mathcal{R}$ we can derive

$$\begin{aligned}
\frac{p_x(r)}{p_y(r)} &= \frac{\left(\frac{\exp(\frac{\epsilon u(x,r)}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(x,r')}{2\Delta u})}\right)}{\left(\frac{\exp(\frac{\epsilon u(y,r)}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(y,r')}{2\Delta u})}\right)} \\
&= \left(\frac{\exp(\frac{\epsilon u(x,r)}{2\Delta u})}{\exp(\frac{\epsilon u(y,r)}{2\Delta u})}\right) \cdot \left(\frac{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(y,r)}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(x,r)}{2\Delta u})}\right) \\
&= \exp\left(\frac{\epsilon(u(x,r') - u(y,r'))}{2\Delta u}\right) \cdot \left(\frac{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(y,r)}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(x,r)}{2\Delta u})}\right) \\
&\leq \exp\left(\frac{\epsilon}{2}\right) \cdot \exp\left(\frac{\epsilon}{2}\right) \cdot \left(\frac{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(x,r)}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\epsilon u(x,r)}{2\Delta u})}\right) \\
&= \exp(\epsilon)
\end{aligned}$$

By symmetry, it follows that $\frac{p_y(r)}{p_x(r)} \geq \exp(-\epsilon)$. □

B. User Interface

This section gives a face to SENTINEL, and showcases several user flows by which providers and consumers can interact with the platform. Providers and consumers alike are initially met with a login page, where they can enter an email address and have a soft wallet (described in section 5.3) generated for them.


Share
Securely share data with others
Empower
Enable your partners to do more with your data without compromising privacy or compliance
Audit
Get an auditable and immutable record of how your data is used
Welcome
Enter your email
LET'S GO!
ABOUT US
BLOG
TERMS OF USE
PRIVACY POLICY
© 2019 Oasis Labs
Share
Securely share data with others
Empower
Enable your partners to do more with your data without compromising privacy or compliance
Audit
Get an auditable and immutable record of how your data is used
Welcome
ABOUT US
BLOG
TERMS OF USE
PRIVACY POLICY
© 2019 Oasis Labs

The Oasis Data Dashboard lets you store data securely and manage who and what you share with others.

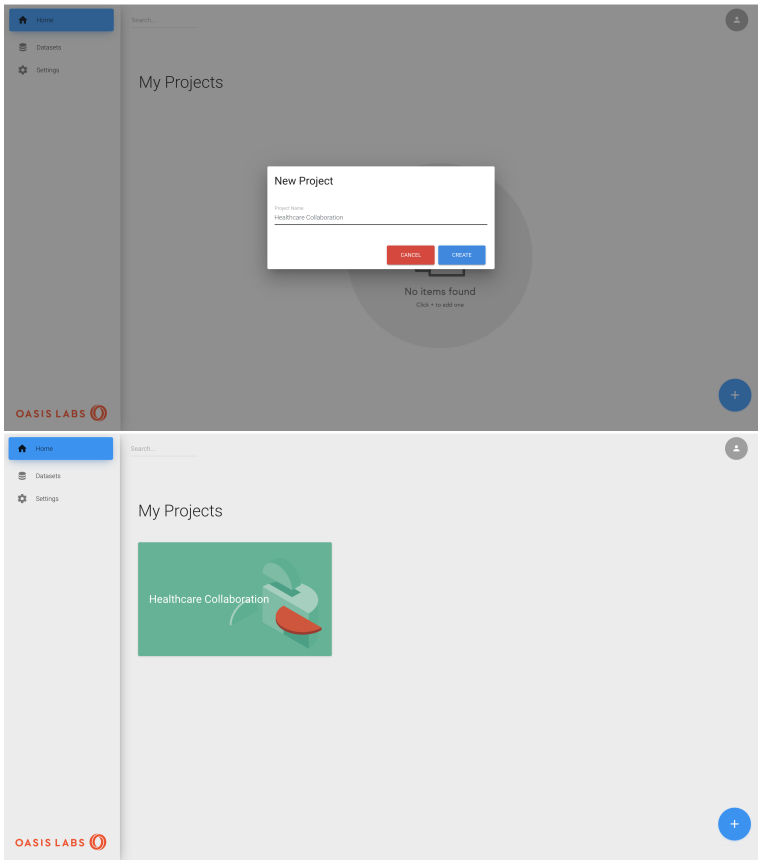
Enter your email below to get started.

The Oasis Data Dashboard lets you store data securely and manage who and what you share with others.

Logging in. This may take up to 1 minute.

Figure B.1.: Login page and wallet generation

Provider `provider@oasislabs.com` has now successfully logged into SENTINEL, and is taken to a landing page, where they can create new collaborative projects.



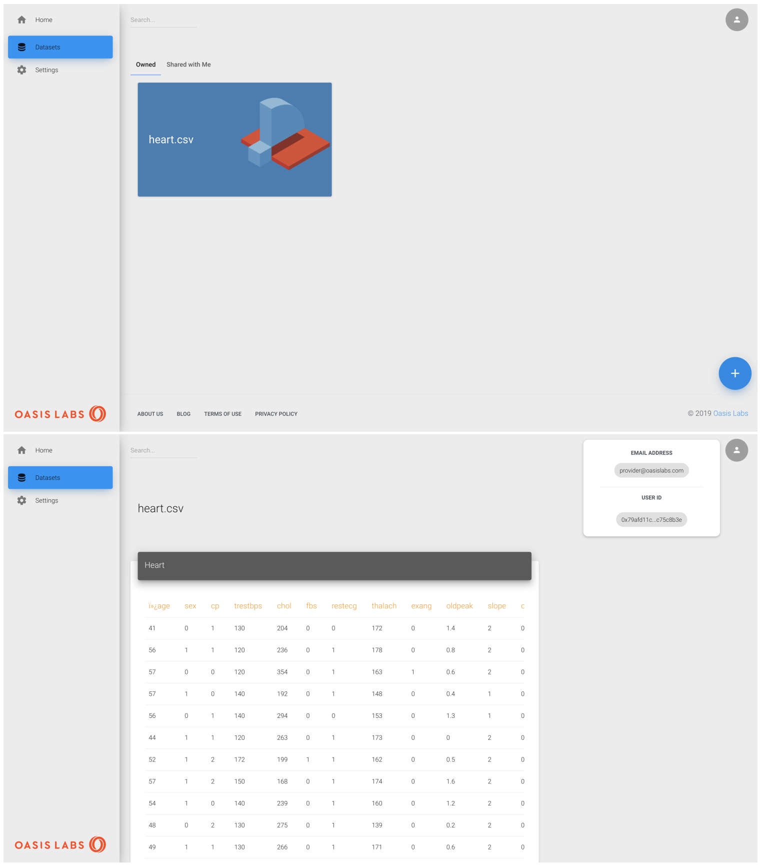
New Project

Project Name
Healthcare Collaboration

No items found
Click + to add one

Figure B.2.: Create a collaborative project

The provider now needs some datasets to add to this project, which they can do by visiting the `Datasets` tab and uploading one. Here, the provider uploads `heart.csv` - the UCI Heart Disease Dataset. They can open and view the dataset directly through SENTINEL, as shown below.



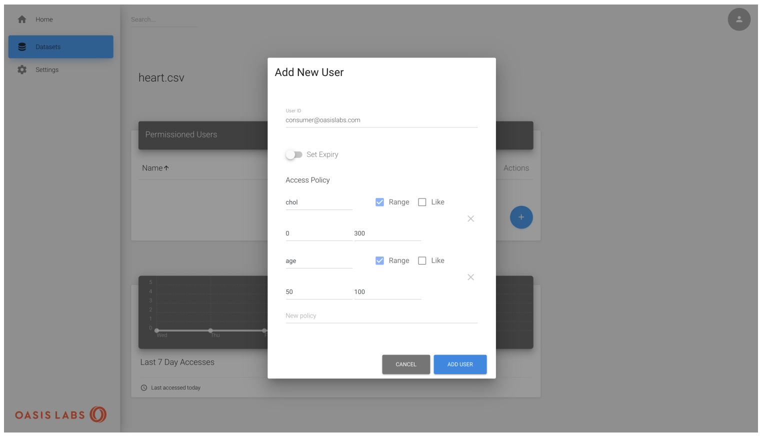
heart.csv

Heart

ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	c
41	0	1	130	204	0	0	172	0	1.4	2	0
56	1	1	120	236	0	1	178	0	0.8	2	0
57	0	0	120	354	0	1	163	1	0.6	2	0
57	1	0	140	192	0	1	148	0	0.4	1	0
56	0	1	140	294	0	0	153	0	1.3	1	0
44	1	1	120	263	0	1	173	0	0	2	0
52	1	2	172	199	1	1	162	0	0.5	2	0
57	1	2	150	168	0	1	174	0	1.6	2	0
54	1	0	140	239	0	1	160	0	1.2	2	0
48	0	2	130	275	0	1	139	0	0.2	2	0
49	1	1	130	266	0	1	171	0	0.6	2	0

Figure B.3.: Upload and view a dataset

The provider can share their dataset with another user, consumer `consumer@oasislabs.com`. We have implemented some stock *range* and *string match* policies for providers to use, and here the provider applies 2 range policies that together act to support Provider 2's policy from section 6.3. Specifically, patients with a cholesterol level above 300 mg/dL and with an age less than 50 are not made visible to the consumer.



heart.csv

Add New User

User ID
consumer@oasislabs.com

Set Expiry

Access Policy

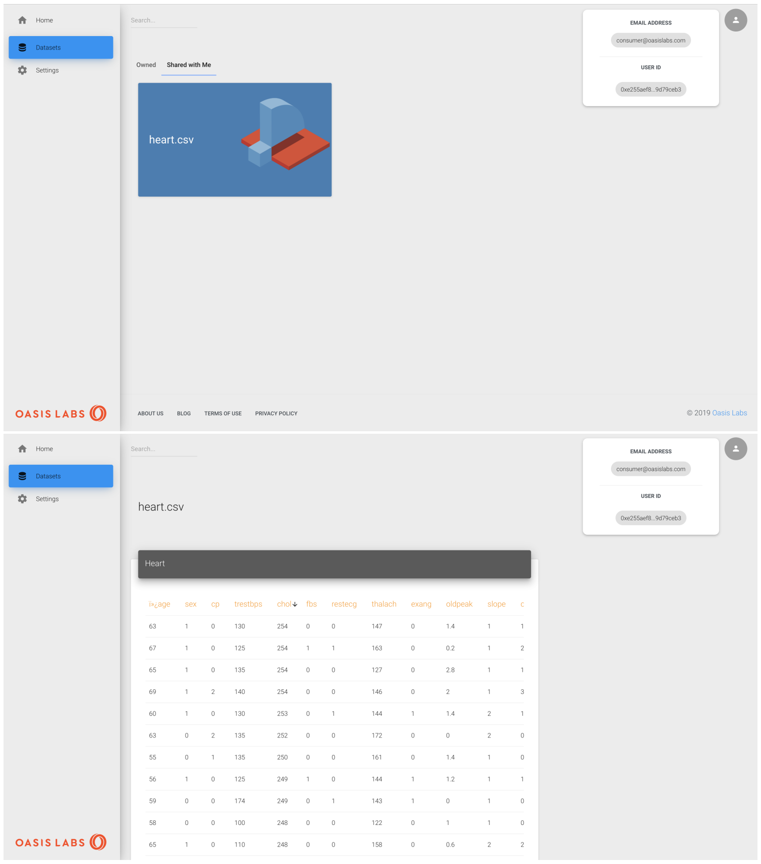
chol [✓ Range] Like
0 300

age [✓ Range] Like
50 100

New policy

Figure B.4.: Share a dataset with enforce policies

Consumer `consumer@oasislabs.com` can now view `heart.csv` in their `Shared Datasets` tab. However, the only rows of the table returned to them are those where the policy has been enforced.



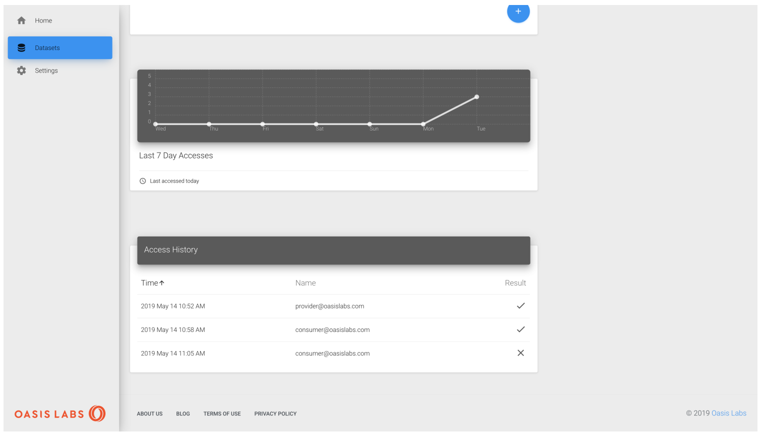
heart.csv

Heart

ï»¿age	sex	cp	trestbps	chol↓	fbs	restecg	thalach	exang	oldpeak	slope	c
63	1	0	130	254	0	0	147	0	1.4	1	1
67	1	0	125	254	1	1	163	0	0.2	1	2
65	1	0	135	254	0	0	127	0	2.8	1	1
69	1	2	140	254	0	0	146	0	2	1	3
60	1	0	130	253	0	1	144	1	1.4	2	1
63	0	2	135	252	0	0	172	0	0	2	0
55	0	1	135	250	0	0	161	0	1.4	1	0
56	1	0	125	249	1	0	144	1	1.2	1	1
59	0	0	174	249	0	1	143	1	0	1	0
58	0	0	100	248	0	0	122	0	1	1	0
65	1	0	110	248	0	0	158	0	0.6	2	2

Figure B.5.: View a shared dataset as a consumer

The provider `provider@oasislabs.com` can now visit the management console for their `heart.csv` dataset, and view an audit log of interactions with their dataset. Below, they see their access to the dataset, the consumer's successful access to the dataset, and a failed access after access revocation.



Last 7 Day Accesses

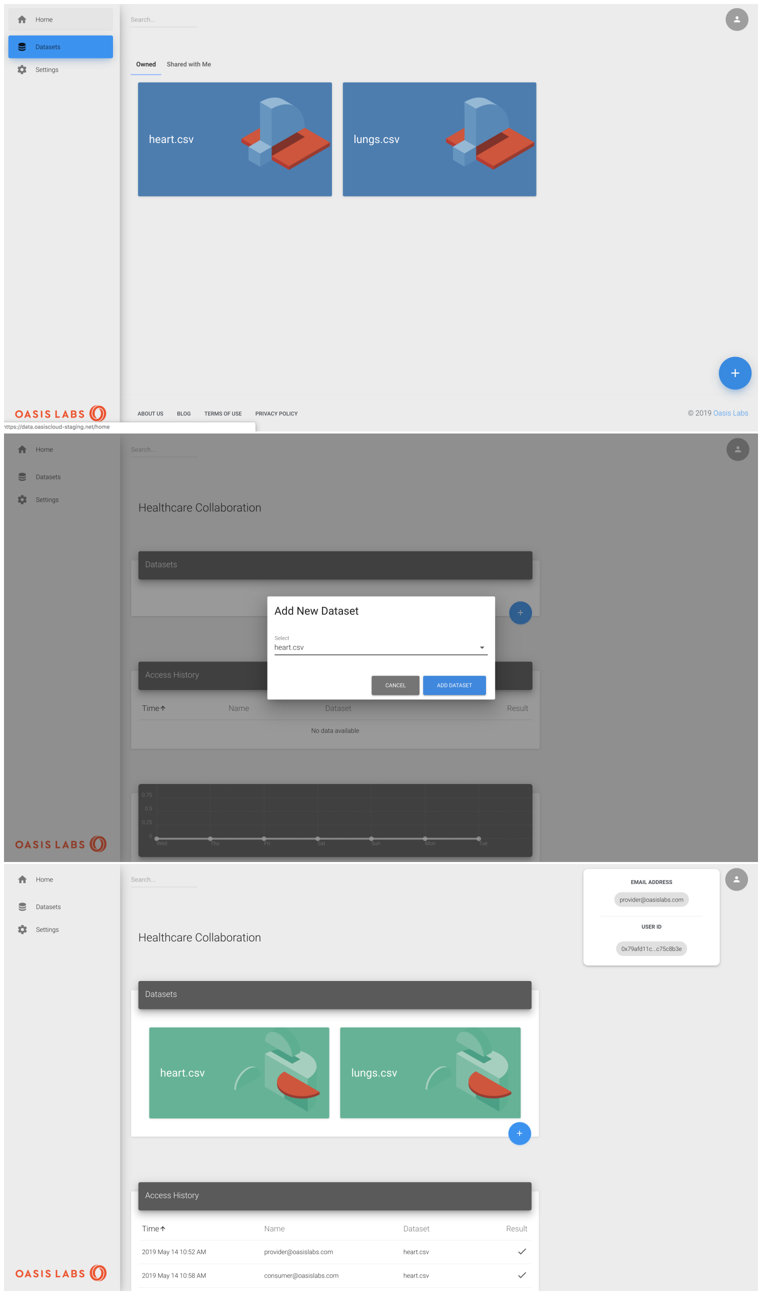
Last accessed today

Access History

Time↑	Name	Result
2019 May 14 10:52 AM	provider@oasislabs.com	✓
2019 May 14 10:58 AM	consumer@oasislabs.com	✓
2019 May 14 11:05 AM	consumer@oasislabs.com	✕

Figure B.6.: Audit a dataset

Finally, the provider can upload a second dataset and add them both to their created project `Healthcare Collaboration`. Audit logs for all added datasets can be viewed in aggregate and other permissioned users can link their policy-protected datasets.



Healthcare Collaboration

EMAIL ADDRESS
provider@oasislabs.com

USER ID
0x79afd11c...c75c8b3e

Access History

Time↑	Name	Dataset	Result
2019 May 14 10:52 AM	provider@oasislabs.com	heart.csv	✓
2019 May 14 10:58 AM	consumer@oasislabs.com	heart.csv	✓

Figure B.7.: Collaborative data sharing