iDASH Privacy & Security Workshop 2021 - Secure Genome Analysis Competition
Track 1 Data Sharing Consent for Health-Related Data Using Contracts on Blockchain

Readme for Data and Skeleton Code

Data Format

The training dataset contains four files, each with 10,000 rows using the following JSON format:

- patient id
- study id
- record time
- patient preferences

The patient id and study id are integers. The record time is a Unix timestamp (in milliseconds). The patient preferences array contains seven binary values:

- demographics
- mental health
- biospecimen
- family history
- genetic
- general clinical information
- sexual and reproductive health

The testing data will be in the same format. However, the test data may contain a different number of records or a different range of the patient id, study id and record time.

Example data (2 out of 10,000 rows used in the training data):

{"patient_id":1675,"study_id":10,"record_time":1620315008846,"patient_preferences":{"DEMOGRAPHICS":true,"MENTAL_HEALTH":true,"BIOSPECIMEN":false,"FAMILY_HISTORY":true,"GENETIC":true,"GENERAL_CLINICAL_INFORMATION":false,"SEXUAL_AND_REPRODUCTIVE_HEALTH":true}}

{"patient_id":1612,"study_id":1,"record_time":1620315016924,"patient_preferences":{"DEMOGRAPHICS":true,"MENTAL_HEALTH":false,"BIOSPECIMEN":false,"FAMILY_HISTORY":false,"GENETIC":false,"GENERAL_CLINICAL_INFORMATION":true,"SEXUAL_AND_REPRODUCTIVE_HEALTH":true}}

Note for Skeleton Code

We have provided a skeleton contract named PatientSharingPreferenceRepo.sol which contains empty functions which must be completed. All testing will be through interaction with these functions. Participants are not limited to only using these functions and may create other functions as well. The only restriction in the participant's design is that each of the initial functions must maintain the provided function names, input, and output formats.

Deployment of the participant's contract will be handled by the testers. The submission should contain only a single file of uncompiled solidity code. We will compile it ourselves using solc 8.4. For further information on the specifics of the code requirements, please refer to the comments in GeneDrugRepo.sol. For initial development we recommend beginning by using Remix (remix.ethereum.org).

The contract must, at a minimum, have the following two public functions to enable the insertion and the query of the data:

addPreferences adds all preferences for one patientId and one studyId at a time as an array of strings (preference names) and a corresponding array of bool (preference values).

Example: For patientId 2, studyId 4 add preference ["DEMOGRAPHICS", "MENTAL_HEALTH", "BIOSPECIMEN"] as [true, true, false] and record time.

Signature:

```
function addPreferences(
        uint _patientId,
        uint _studyId,
        uint256 _recordTime,
        string[] memory _preferenceNames,
        bool[] memory _preferenceValues) public
)
```

getConsentingPatientIds takes a studyId and an array of preference names and returns all patientIds that have consented to all preference names in the list.

Example: For study 8 return an array of patientIds that return true for all three preference names in array ["DEMOGRAPHICS", "MENTAL_HEALTH", "BIOSPECIMEN"].

Signature:

```
function getConsentingPatientIds(
        uint _studyId,
        string[] memory _requestedSitePreferences) public view returns (uint[] memory)
)
```

Additional Rules of Solution

The solution should be designed in such a way as to be optimized for reading and storing live changes to a data sharing system, instead of compressing and handling many records at once. Therefore, the solution may be tested by sending batches of records (e.g., per 1000 records , or even per 1 record) at a time. To simulate a real-world data sharing solution, any sort of buffering method (which may lead to a loss of data) is not allowed.

Also, the solution should be able to store the data on one node of the blockchain network and retrieve on any other node, without any other method of communication between machines. The solution should be symmetric (i.e., identical software for each node). No other computing/storage resources will be available except the testing virtual machines. Furthermore, the single file smart contract solution should be standalone, such that upon being copied it should be able to read and write to an existing blockchain without any external setup or configuration, no calls to outside resources.