README.md

Protocols for Private STR Queries

This library contains a proof-of-concept implementation of our protocol for performing private STR queries. This implementation is a research prototype and is not intended for use in production environments.

Paper Reference

This is an implementation of the protocols from the following paper:

- Jacob A. Blindenbach, Karthik A. Jagadeesh, Gill Bejerano, and David J. Wu. Avoiding genetic racial profiling in criminal DNA profile databases. A preliminary version of this work is available on bioRxiv.

Please contact Jacob Blindenbach (jab7dq@virginia.edu) if you have questions about this code.

Github Repository

An updated version of the code is maintained in the `private-codis` Github repository.

System Requirements

The provided code was tested on `Ubuntu 18.04 LTS` and was tested with the following compiler and interpreters:

- `python 3.6.9`
- `clang 6.0.0 (x86-64)`

There are no external dependencies other than the standard Python/C++ libraries. The implementation relies on the Intel AES-NI instruction set, which is supported by most modern Intel processors (since the Westmere architecture).

Compiling the Code

To compile the code, simply run `make` in the project root directory.

Local Benchmark

After compiling the code, to run a local benchmark, use the following command:

```
python3 run.py -N DATABASE_SIZE -config CONFIG_FILE -in IN_DATABASE -p PORT -local LOCAL
```

- **DATABASE_SIZE:** Specifies the size of the database the secure query will be performed on.
- **CONFIG_FILE:** Specifies the STR configuration. For a list of configurations supported by this sample implementation, see the `configs` directory.

- **IN_DATABASE (optional parameter):** Specifies if the simulated query will be in the database or not. 0 denotes the record is not in the database, while 1 denotes that the record is in the database (defaults to 1).
- **PORT (optional parameter):** Specifies port used for client/server communication (defaults to 8000).
- **LOCAL (optional parameter):** Specifies whether to run a local benchmark (client/server on the same machine) or a networked benchmark (defaults to 1). If a networked benchmark is requested, the script will generate the data files and the OT correlations for the client and the server. Refer to the section on Client/Server Benchmarking for further details on running the protocol over the network.

Example

```
python3 run.py -N 100000 -config ./configs/US-Current.config -in 1
```

This script will generate a synthetic database with 100000 STR records according to the current US standard. It will also generate a sample query that matches one of the records within the database. The script will then start a local client and server instance to execute the protocol (using the default port 8000 for all communication).

Example Output

Sample output for an example run of the above script:

```
python3 run.py -N 100000 -config ./configs/US-Current.config -in 1

Collecting protocol parameters
Generating test case
Number of OT correlations: 11600000
Query is in database at location: 24520
Running local test
[client] successfully connected to server
[client] starting client protocol
[client] found a match at index 24520
[client] protocol execution completed
protocol execution time: 0.848771 seconds
total bytes sent: 4475000
total bytes received: 13600000
total bytes communicated (received + sent): 18075000
```

Note that the data generation script generates a *random* database and query, so the exact parameters will vary from run to run.

Client/Server Benchmarking

To run a benchmark in a networked environment with separate client/server instances, proceed as follows:

1. **Preprocessing:** First, run the `run.py` script with the local flag turned off:

   ```
   python3 run.py -N DATABASE_SIZE -config CONFIG_FILE -in IN_DATABASE -local 0
   ```

2. Copy the `data/client` directory to the client instance and the `data/server` directory to the server instance. The `data/client` directory contain the precomputed client OT correlations as well as the query while the `data/server` directory contain the precomputed server OT correlations and the database.
3. Copy the `data/info.txt` file to the client and server instance. This file contains the protocol arguments (database size and STR configuration information) that will be used during execution
4. **Server:** Start the server by running on the server

```
./tests/ProtocolServer INFO_FILE SERVER_DATA_DIR PORT
```

In this case server will start listening on the specified `PORT`. For example,

```
./tests/ProtocolServer ./data/info.txt ./data/server/ 8000
```

5. **Client:** Start the client using

```
./tests/ProtocolClient INFO_FILE CLIENT_DATA_DIR SERVER_IP_ADDDRESS PORT
```

The client will connect to the server at address `SERVER_IP_ADDDRESS` using `PORT`. Once the server accepts the connection, the client and server will initiate the protocol using the data in the `data/client` and `data/server` directories, respectively. For example,

```
./tests/ProtocolClient ./data/info.txt ./data/client/ 127.0.0.1 8000
```

Example Output

Sample output for an example run of the preprocessing, server, and the client:

Preprocessing

```
python3 run.py -N 100000 -config ./configs/US-Current.config -in 1 -local 0

Collecting protocol parameters
Generating test case
Number of OT correlations: 11600000
Query is in database at location: 82837
```

Server

```
./tests/ProtocolServer ./data/info.txt ./data/server/ 8000

[server] finished loading data from disk
[server] accepted connection from client
[server] starting server protocol
[server] protocol execution completed

bytes sent: 13600000
bytes received: 4475000
```

Client

```
./tests/ProtocolClient ./data/info.txt ./data/client/ 127.0.0.1 8000

[client] successfully connected to server
[client] starting client protocol
[client] found a match at index 82837
[client] protocol execution completed
protocol execution time: 0.873334 seconds
total bytes sent: 4475000
total bytes received: 13600000
total bytes communicated (received + sent): 18075000
```

Note that the data generation script generates a *random* database and query, so the exact parameters will vary from run to run.

Sample Benchmarks

We provide sample benchmarks of our protocol below (using the provided implementation). These measurements were collected by running the protocol on two Amazon EC2 instance (`m4.2xlarge`), with the client instance located in the Northern California region (`us-west-1`) and the server instance located in the Northern Virginia region (`us-east-1`). The network latency between the two instances was roughly 60ms and the bandwith was roughly 25 MB/s.

Protocol Execution Time and Communication Cost

The following table summarizes the average protocol execution time and network communication for running the protocol over a database of 1,000,000 entries and different STR configurations (described in the `configs` directory). Each measurement was taken using the exact procedure described in described in Client/Server Benchmarking with the respective configuration. We refer to the paper for more details on each configuration.

CODIS System	Configuration	Protocol Execution Time (s)	Communication (MB)
US System, current	`configs/US-Current.config`	38.35	172
US System, pre-2017	`configs/US-Old.config`	27.74	115
UK-like System	`configs/UK.config`	24.96	101
EU-like System	`configs/Europe.config`	32.66	143
Chinese-like System	`configs/China.config`	39.95	180
NIST 40 Loci System	`configs/US-New.config`	81.62	389

Precomputation Size and Number of OT Correlations

The following table summarizes the size of the client and server OT correlations as well as the total number of OT correlations for running the protocol over a database of 1,000,000 entries and different STR configurations (same as above). As before, each measurement was taken using the exact procedure described in described in Client/Server Benchmarking with the respective configuration.

CODIS System	Client Precomputation (MB)	Server Precomputation (MB)	Number of OT Correlations
US System, current	58	130	116 million
US System, pre-2017	39	87	78 million
UK-like System	35	76	69 million
EU-like System	48	108	97 million
Chinese-like System	61	136	122 million
NIST 40 Loci System	132	292	266 million

Full Benchmarks

We include additional benchmarks for CODIS databases of different sizes for each of the configurations in the `benchmarks` directory. We refer to the paper for a full discussion of the provided values.