Lens: A Service-Oriented Platform for Privacy-Preserving Computation over Personal Data

Group #12: Shomil Jain (CS 262A/CS294-171), Ben Hoberman (CS 294-171), Solomon Joseph (CS 294-171)

Motivation

Google, Apple, Facebook, and other third-party services **collect, store, and operate on large quantities of personal data**. These companies most frequently use this data for internal personalization, advertising, or financially-motivated purposes (e.g. selling to data brokers). However, when they expose this data to third-party application providers, **they expose raw, plaintext user data to application providers**, and force users to trust these third parties with data access.

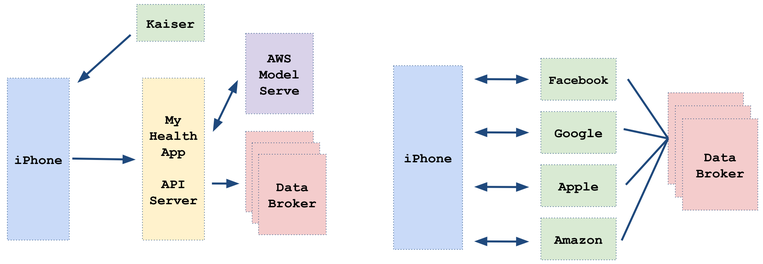

Figure 1: Limitations of the Current Paradigm of Computation over Personal Data.
An example of a health application that performs computation on personal data (left), and the data silos created by multiple third parties on the right.

Background

Confidential computing systems usually fall into three categories: hardware-based (e.g. secure enclaves), cryptography-based (e.g. homomorphic encryption), and hybrid systems. We chose to explore secure hardware for this project for reasons concerning practicality, rapid adoption (e.g. "lift and shift"), and availability of SGX-enabled devices on common cloud providers (e.g. Azure).

The Lens framework builds on top of Ego, which consists of a modified Go compiler, additional tooling (e.g. a tool to sign enclave code), and a Go library (e.g. for remote attestation). Go compiler-generated syscalls are replaced with Enclave libc calls - Open Enclave's libc calls. All other instructions are run normally. Because Ego compiles for an enclave instead of supporting arbitrary code running in an Enclave, Ego - at its core - is much simpler than Haven/SCONE. Ego is an open-source project.

Technical Architecture

Lens is an end-to-end application framework built on top of Ego & Open Enclave that enables users to leverage third party applications that perform secure computation over their personal data.

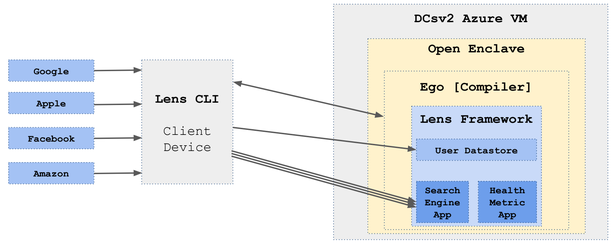

Figure 2: Technical Architecture.

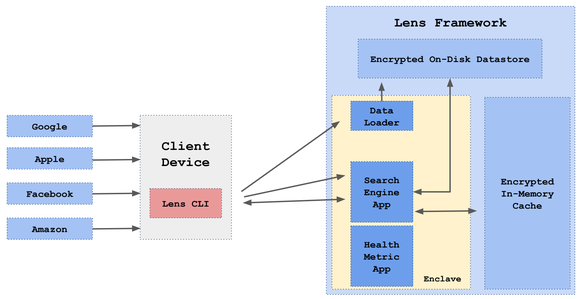

Figure 3: User Flow.
Remote attestation occurs over RA-TLS.

1. **Perform GDPR Data Download Requests.**
   This is something the user has to do. See "Future Directions" for an optimization here.
2. **Perform remote attestation with the enclave.**
   This assumes the CLI knows the enclave's current measurement (e.g. so the client can verify that the expected code is running in the enclave). The attestation service is baked into Ego's Go library.
3. **Upload data to the Lens Framework through the Lens CLI.**
   Files are compressed and streamed into the enclave. The data loader stores these on disk (encrypted).
4. **To initialize a module, initiate the Preprocessing command.**
   Some modules require preprocessing (e.g. a search engine requires building a search index ahead of query-time). Others don't.
5. **To make a query, initiate the Query command.**
   Modules can expose arbitrary queries to users, and users may pass arbitrary parameters for modules to process. For example, the Search application exposes a *search* route that takes in a *keyword* parameter. If the user enters *cs161*, then a joint search is performed over the in-memory search index over that user's documents, and all resulting matches are streamed to the user.

Technical Contributions

Our main technical contributions include –

1. An end-to-end application framework for confidential computing over personal data
2. An extension to this framework enabling secure multi-party computation, where users can perform a joint computation over the synthesis of their data
3. A application-level module for Ego enabling secure disk I/O
4. Two example applications leveraging Lens
   a. A confidential search index over personal data
   b. A confidential multi-party health leaderboard application

Results

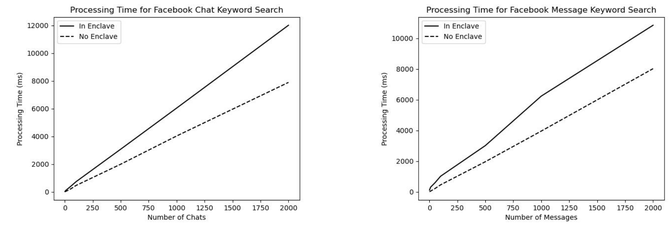

Figure 4: End-to-End Latency Results for the Search Application/
We note the percent-impact on performance when we compare our "secure" computation system to an insecure alternative (one that runs outside of the enclave with no encryption and all processing done over plaintext). The "secure" mode is, on average, around 15% slower than the insecure - which we believe is a reasonable performance dip for the privacy and security advantages.

Note: microbenchmarks will be included in our final paper.

Future Directions

- Fill remaining holes in Ego (e.g. integrate transparent encrypted disk I/O into compiler-level, instead of at application-level)
- Add real-time support architecture (e.g. incrementally poll updates from API's, instead of using data archives)
- Add support for multi-service applications (e.g. through Kubernetes service mesh)
- Strengthen the threat model to **eliminate access pattern attacks**
- Enable managing, deploying, and serving **untrusted applications with proprietary code** in a secure fashion (e.g. a potentially-malicious piece of TurboTax code with proprietary model weights)
- Build out more interesting applications for more comprehensive design consideration evaluation (e.g. integrate 100's of data sources, instead of just two)