Fully Decentralized Information Flow Control for Cloud Object Stores

Qiao Kang[†], Ang Chen[†], Ennan Zhai[*]

[†]Rice University, [*]Alibaba Groups

Decentralized information flow control (DIFC) is an approach to ensuring data confidentiality. In DIFC systems, both data and principals (e.g., processes) are assigned with security labels. A trusted policy enforcer then tracks the labels and checks whether a particular data flow should be allowed or not. At the core of DIFC is an egalitarian mechanism for label creation and delegation, where each principal can create its own labels and assign them to different data sources. This eliminates the "superuser" role from and decentralizes trust in a traditional IFC system.

One challenge that arises in a DIFC system is the need for data aggregation across labels. Consider an email service, such as Gmail: the business model mandates that the service provider needs to be able to query aggregate data from all or a subset of its users. The querier (in this case Google) can then use the aggregate data to gain business benefits or to provide better service. Strictly enforcing DIFC in this scenario would eliminate this possibility, as each user only keeps data to itself by labeling it. Existing DIFC systems solve this problem by creating special components in the system that can accept data with different labels and declassify the aggregate results [2]. For instance, Google might create a data aggregation engine that has access to all users' data, and can release the results of the analysis to the querier by declassifying these labels. This effectively brings back a "superuser" role in the otherwise fully decentralized trust model.

Our goal: Fully decentralized trust. We propose FullStar, a *fully* decentralized IFC system that preserves the ability for data aggregation without requiring a privileged component. FullStar is designed as a DIFC-enabled distributed object store, and intended to be integrated with OpenStack Swift as the storage service for cloud tenants. Our key insight is that we can *separate individual user data from attributes that require aggregation, and protect them differently*. We call the former `userData` and the latter `aggData`. We leverage traditional DIFC to protect `userData`, by tainting the data objects and all principals that access them. For `aggData`, however, we protect it with cryptographic mechanisms that enable aggregation without allowing the aggregation service to see the individual data items. Such an aggregation service does not learn anything about individual user's data, so it never gets tainted with any label. A trusted DIFC proxy (e.g., the OpenStack Swift proxy server) interposes between the object store and the aggregation engine, and decides to either add DIFC labels to user data, or encrypt individual attributes for aggregation. As such, FullStar eliminates the privileged aggregation component that can declassify all users' labels, achieving a fully decentralized IFC system.

FullStar encrypts `aggData` using homomorphic encryption algorithms to preserve computability. Homomorphic cryptosystems can enable computation over encrypted data. It has been shown that statistical analysis queries, such as computing data variance or skew, can be supported using advanced encryption mechanisms [1]. Aggregation engines can perform computation over encrypted data and send the results to the querier for decryption. It may also be possible to support SQL-like queries over encrypted data by combining multiple cryptographic schemes, just like in CryptDB.

Example: Email service. As a concrete example, consider an email service that wants to collect aggregate data for analysis. The querier might issue a query that asks for the average attachment size for a particular group of users. Here, all emails and attachments are tainted with per-user DIFC labels, and FullStar does not directly send such data to the aggregation service. Rather, it encrypts just the attributes the querier asks about (i.e., attachment sizes), and lets the aggregation engine perform additions over all encrypted attributes without revealing the plaintext to it. The encryption keys can be negotiated by the querier and FullStar, and never get released to the aggregation engine. Therefore, we preserve the ability to perform aggregation without having to create a superuser role in the system.

Open questions. As ongoing work, we are exploring several open questions. In terms of the *threat model*, FullStar relies on the assumption that the querier and aggregation engine do not collude to decrypt user data. We would like to further investigate the possibility of relaxing this assumption. In terms of *frequency of aggregation*, homomorphic encryption may result in high overhead, so it is important to cache computed results so that they can be reused later. We are also investigating the challenges and opportunities in a *multi-tenant environment*, and working torwards a *prototype* that can be integrated with OpenStack Swift and support a range of practical use cases.

References

[1] N. Neha, V. Willy, and M. Virza. zkLedger : Privacy-preserving auditing for distributed ledgers. In *Proc. NSDI*, 2018.

[2] D. Schultz and B. Liskov. IFDB: Decentralized information flow control for databases. In *Proc. EuroSys*, 2013.

Qiao Kang: Student author

Fully Decentralized Information Flow Control for Cloud Object Stores

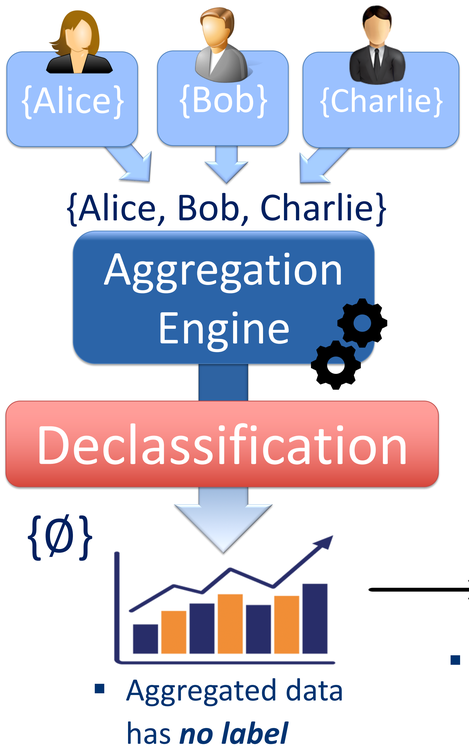
Qiao Kang[†], Ang Chen[†], Ennan Zhai[*]

[†]Rice University, [*]Alibaba Group

Email: qiaokang@rice.edu

1. Problem

- ***Background:*** Decentralized Information Flow Control (DIFC) systems taint data and control its flow by adding and tracking labels (e.g., DStar, IFDB).
- ***Problem:*** Cross-label data aggregation requires tainting the aggregation engine with ***all user labels***.
  - Aggregated data cannot flow out of the engine.
- ***Existing Solution:*** Grant a "superuser" privilege to the aggregation engine, so that it can declassify all user labels.
  - However, if the aggregation engine is compromised, all security would break down.



- Confidential data protected by **per-user labels**
- ***Tainting*** the aggregation engine with all user labels
- ***Declassifying*** all user labels
- Aggregated data has ***no label***
- Aggregated can be ***shared*** freely

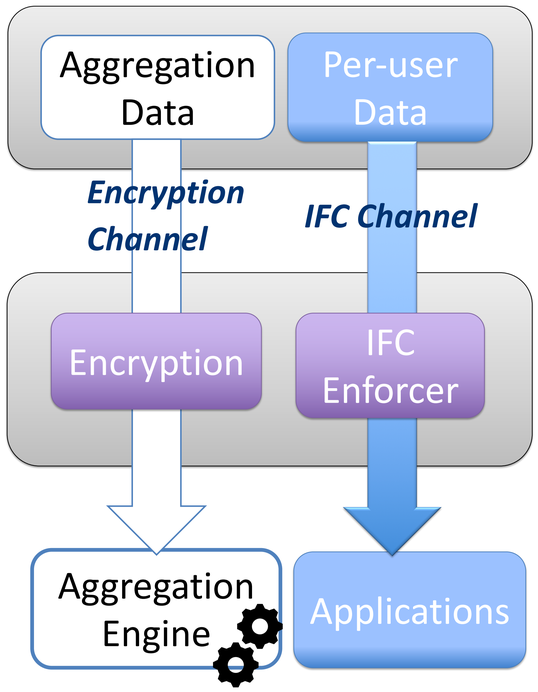
- ***Goal:*** Achieve ***Fully Decentralized Information Flow Control*** by eliminating the aggregation engine with superuser privilege.

2. Our Approach

- ***Key Insight****: We* can separate per-user data from aggregation data (i.e., attributes to perform aggregation over), and protect them with different security mechanisms.

  (1) ***IFC channel*** taints per-user data with IFC labels.
  (2) ***Encryption channel*** encrypts aggregation data before sending to aggregation engines.

- ***Our System: A Cloud Object Store with Fully Decentralized Information Flow Control***

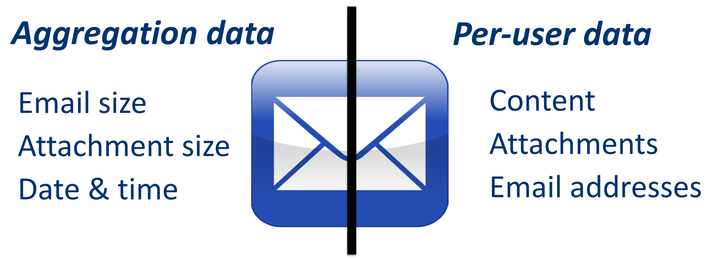


- ***Object storage layer*** differentiates between the two types of data
- ***Proxy layer*** decides whether to add IFC labels or perform encryption
- Only ***applications*** with access to per-user data would be labeled

- ***Benefit: No single application can declassify all user labels***.
  - Removes the aggregation engine with superuser privilege
- ***Challenge: Preserving computability of encrypted data***
  - Tentative solution: homomorphic encryption.

3. Example

- ***Example***: Email service.
  - The service provider may issue periodic queries for statistical analysis, for instance, average attachment size for a particular group of users in this month.
  - For this purpose, the attachment size attributes are protected as aggregation data in our system.



How to define the boundary in the middle?

- Use user-provided policies to initialize the boundary
- Performance optimizations depending on the access patterns

Which aggregation functions should we support?

- Basic: addition, multiplication, average, variance, etc.
- Advanced: keyword search, SQL query, etc.

4. Ongoing Work

- Our model assumes that queriers do not ***collude*** with aggregation engines to leak users' confidential data. We are investigating the possibility to further relax this assumption.
- Add a ***data cache*** in the encryption channel to reduce the overhead of homomorphic encryption.
- Use ***differential privacy*** as an alternative to homomorphic encryption.

This work is partially supported by the National Science Foundation through the grant CNS-1801884