DeCor: Decorrelating unsubscribed users' data from their identities

Anonymous Authors

Abstract

100-200 words that convince you to read this.

1 Introduction

1.1 Motivation

Web application companies face increasing legal requirements to protect users' data. These requirements pressure companies to properly delete and anonymize users' data when a user requests to *unsubscribe* from the service (i.e., revoke access to their personal data). For example, the GDPR requires that any user data remaining after a user unsubscribes is *decorrelated*, i.e., cannot be (directly or indirectly) used to identify the user [4].

In this paper, we propose DeCor, a new approach to managing user identities in web applications. DeCor meets the de-identification requirements in the GDPR, and goes beyond: with DeCor, it is possible for users to switch between a privacy-preserving unsubscribed mode and an identity-revealing subscribed mode at any time. This facilitates important new web service paradigms, such as users granting a time-limited "lease" of data to a service instead of having a permanent service account.

1.2 Goals

DeCor's goal is to provide the following properties:

Decorrelation. Informally, decorrelation should guarantee that it is impossible to distinguish between two records formerly associated with the same unsubscribed user and two records from different unsubscribed users.

Resubscription. Users should be able to easily switch between a privacy-preserving unsubscribed mode and an identity-revealing subscribed mode, without permanently losing their application data.

DeCor must implement these properties while ensuring (1) performance comparable to today's widely-used databases, and (2) easy adoption (requiring little to no modification of application schemas or semantics).

1.3 Threat Model

We make the following assumptions of an attacker whose goal is to break decorrelation:

- An attacker can perform only those queries allowed by the application API: an attacker can access the application only via its public interface. [lyt: Alternatively, an attacker could perform arbitrary queries on some public subset of the application schema (e.g., all tables other than the mapping table, or all tables marked with some compliance policy); arbitrary queries over the entirety of the table are out of scope, unless "private" tables are removed and stored by unsubscribing users.]

- An attacker cannot perform application queries to the past or search web archives: information from prior application snapshots may reveal exactly how data records were decorrelated from unsubscribed users.

- An attacker cannot gain identifying information from arbitrary user-generated content (for example, a reposted screenshot, or text in user posts or comments). Decorrelation seeks to remove identifying information from user-generated data that can be enumerated or follows a specific pattern (e.g. a birthday or email address), and application metadata (e.g., date of postings, database ID columns).

1.4 A General Model for Decorrelation

Application data is structured as tables, each containing a different *data entity*, e.g., a post, user, or vote. Entities can either be externally added via the application API by a client,

or internally added by the application (e.g., tags or application metadata). Queries write, read from, and compute over entities. As entities are processed by a query, their path from table to application client can be represented as *flows* through computations specified by the query: each computation can be classified as either a join, an aggregation (e.g., count, min, max), or a set operation (e.g., union, intersection) between flows.

When a user U unsubscribes, decorrelation must (1) determine which entities may reveal identifying information about U, and (2) prevent queries from leaking this information.

To address (1), the programmer developer specifies identifying tables or table columns via developer annotations on the application schema: tables or table columns can be marked as identity-sensitive. If only tables are marked, all columns are assumed to be sensitive.

To address (2), we model queries as a dataflow computation, and perform actions on sensitive data flowing into computation nodes (joins, aggregations, or set operations) to prevent the computation result from exposing identifying information. Examples of actions include:

- Deleting sensitive entities, or removing sensitive entities from the dataflow. Pros: achieves leave-no-trace. Cons: can disrupt application semantics, and removes any useful information from an unsubscribed user.
- Anonymizing sensitive entity columns: integer or string fields can be randomized, or generated from a distribution of field values (e.g., a phone number).
- Injecting fake entities (noise) or fake associations (changing foreign key values).

[lyt: This seems similar to IFC declassification in a DP/probabilistic sense?]

Note that only certain computations may leak information, and certain actions may need to take place on only one of the flows into the computation instead of both to prevent sensitive information leakage. Here, we list all possible scenarios, and which actions should be taken on which entities flowing into the computation:

- Aggregation/pipelined computations: depend on only one flow of entities. If the entities or entity columns are sensitive, the result is also considered sensitive, and should be acted upon appropriately before reaching the end user.
- JOINs: if one or both of the flows into the joins include sensitive entities or entity columns, all results of the JOIN depend on sensitive entities and should be acted upon before reaching the end user.
- Unions: only the sensitive entities or entity columns must be acted upon in the resulting union
- Intersection: if one of the flows is sensitive, the resulting flow must be acted upon appropriately
- Minus: if either the entities in the subtrahend or the entities in the minuend are sensitive or have sensitive columns, the resulting set of entities must be acted upon. Information can leak about the entities which have been subtracted, or about the entities which remain.

1.5 DeCor: Instantiating the Decorrelation Model

DeCor is one potential architecture for implementing this model. In DeCor, sensitive columns are user ID fields (annotated by the application developer). These user IDs are numerical user keys uid_{key} that are each unique to one user, and map entities to a particular user. Entities containing multiple uid_{key}s are considered shared among the identified users.

DeCor handles flows that contain sensitive entities belonging to unsubscribed users by anonymizing the uid_{key} column: the uid_{key} is replaced by a unique ghost ID (gid_{key}) per entity belonging to the unsubscribed user. By replacing uid_{key} values with gid_{key} values, DeCor ensures that the computations on the dataflow decorrelate these entities with the unsubscribed user: an unsubscribed user's data is split into individual pieces.

While the model implies that actions are taken at runtime (when entities pass through query operators), DeCor implements actions on uid_{key}s by storing the gid_{key}s in the underlying application data tables, saving a mapping of gid_{key}s to uid_{key} in the database, and maintaining materialized views to answer application queries that expose real or ghost identifiers depending on whether a particular user is subscribed. DeCor implements a database shim layer that transparently rewrites application queries to query the materialized view (the "acted-upon" result) rather than the data tables, and which propagates updates appropriately to the materialized view. In essence, the materialized views cache the result of the actions taken after a query operation such as a JOIN.

An alternative action might replace all unsubscribed users' uid_{key}s with one *global placeholder* value, essentially collapsing all unsubscribed users' entities into one pool. However, this erases all user-specific data, making resubscription and subsequent recorrelation of a user's entities with the user's identity difficult. DeCor's ghost identifiers allows users to reactivate their account and undo the decorrelation: uid_{key}s can be linked back to a set of unique gid_{key}s. This gives users the ability to freely unsubscribe to protect their privacy without worrying about losing their accounts. [lyt: Ghosts also make schema changes / changing the location of data records easier to support.]

Furthermore, ghost IDs provide increased decorrelation guarantees: if decorrelation utilizes a global placeholder and entities with uid_{key}s are exposed by application queries, an

attacker can determine with 100% certainty which queried entities are unsubscribed entities, namely any entity with uid_{key} equal to the global placeholder. If only one or a few users have unsubscribed, correlating these unsubscribed entities back to one identity may be trivial.

However, if decorrelation instead relies on ghosts, an attacker cannot guarantee that any revealed identifier is a ghost instead of a real (subscribed) user. Instead, the attacker must calculate the probability that an identifier belongs to a ghost using external knowledge about identifiers (e.g., GIDs may be randomly generated in a identifiable pattern) or other information exposed by the entity. This external, non-uid_{key} information can be either marked sensitive (so DeCor with act upon these entity columns as well) or treated as out of scope.

2 Background and Related Work

Companies have developed frameworks to avoid deletion bugs (e.g., DELF [1] at Facebook), but many applications decorrelate only coarsely by associating remnants of data with a global placeholder for all deleted users, or do not decorrelate at all (e.g., replacing usernames with a pseudonym).

[lyt: (cite Reddit, Lobsters, others?)]

The right to be forgotten has also been formally defined by Garg et al. [2], where correct deletion corresponds to the notion of leave-no-trace: the state of the data collection system after a user requests to be forgotten should be left (nearly) indistinguishable from that where the user never existed to begin with. While DeCor uses a similar comparison, their formalization assumes that users operate independently, and that the centralized data collector prevents one user's data from influencing another's.

Other related works:

- Deceptive Deletions for protecting withdrawn posts: https://arxiv.org/abs/2005.14113
- "My Friend Wanted to Talk About It and I Didn't": Understanding Perceptions of Deletion Privacy in Social Platforms, user survey https://arxiv.org/pdf/2008.11317.pdf; talk about decoy deletion, prescheduled deletion strategies [3]
- Contextual Integrity
- ML Unlearning

3 Design

4 Implementation

5 Evaluation

6 Discussion

Acknowledgments

Availability

[lyt: USENIX program committees give extra points to submissions that are backed by artifacts that are publicly available. If you made your code or data available, it's worth mentioning this fact in a dedicated section.]

References

[1] Katriel Cohn-Gordon, Georgios Damaskinos, Divino Neto, Joshi Cordova, Benoît Reitz, Benjamin Strahs, Daniel Obenshain, Paul Pearce, and Ioannis Papagiannis. DELF: Safeguarding deletion correctness in online social networks. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, August 2020.

[2] Sanjam Garg, Shafi Goldwasser, and Prashant Nalini Vasudevan. Formalizing data deletion in the context of the right to be forgotten. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 373–402. Springer, 2020.

[3] M. Minaei, Mainack Mondal, and A. Kate. "my friend wanted to talk about it and i didn't": Understanding perceptions of deletion privacy in social platforms. *ArXiv*, abs/2008.11317, 2020.

[4] E Parliament. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). *Official Journal of the European Union*, L119(May 2016):1–88, 2016. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC.