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 [2].

In this paper, we propose DeCor, a new approach to managing user identities in web applications. DeCor meets the decorrelation 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 while preserving an application's semantics:

Decorrelation. Informally, ideal decorrelation guarantees 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 (decorrelation should be automated without needing to modify application schemas or semantics).

1.3 Threat Model.

We address applications in which application data consists of *data records* and computations (such as aggregations) that may be performed over these data records. Data records are considered sensitive, private data records when they contain *user identifiers (UIDs)*. For example, the user table ID field, usernames, or phone numbers are all UIDs, and a table row with any of these as columns is a private data record. Data records may also contain *indirect identifiers (IIDs)*, which include application metadata such as the time of posting or category of post.

We assume that all user data records contain a user table ID field, a numerical user key uid_{key} that is unique to each user, and which ties all data records to a particular user. Data records containing multiple uid_{key}s are considered shared data records private to the identified users.

We refer to data records belonging to unsubscribed users as *remnants*.

We make the following assumptions of an attacker whose goal is to break decorrelation and reveal identifying information about data remnants:

- 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 ap-

plication 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 only IIDs (e.g., date of postings, database ID columns), or controllable UIDs (e.g., a birthday or email address) [lyt: There needs to be a clearer definition of what is "arbitrary" and what is "controllable"]

1.4 Decorrelation Guarantees

Decorrelation Techniques. A common strawman decorrelation technique replaces all unsubscribed users' uid_{key}s with one *global placeholder* user, essentially collapsing all unsubscribed users' data into one pool. This means that all remnants can be identified by one distinct gid_{key}. Other UIDs in data records (such as emails, phone numbers) are also replaced by global placeholders (such as a default email address) that are assigned to this global user. Using a global placeholder, however, makes resubscription challenging: users can no identify which unsubscribed data records belong to them if all user-specific data has been erased from the system.

An alternative technique used by DeCor generates a unique *ghost user* for each data remnant, essentially splitting unsubscribed users' data into individual pieces, and replacing one uid_{key} with many gid_{key}s, one per data record owned by the user. Ghost users each have distinct GIDs in place of UIDs (e.g., a randomly generated email address per ghost). Unlike a global placeholder, DeCor 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.]

We next describe how these two techniques can lead to different decorrelation guarantees.

Measuring Decorrelation. We say that the system achieves (Q, ε)**-decorrelation** when the probability that an attacker can correctly determine that any two distinct remnants belong to any one user is less than ε, given information derivable from performing any Q queries. [lyt: Note that the *correctly* is probably essential here: An attacker who will mistakenly correlate two remnants with some user actually faces more challenges]

Defining this probability requires calculating two others:

p_{remnant} : The probability that the attacker can determine that any data record is a remnant

p_{linked} : The probability that the attacker can determine that any two remnants belong to the same individual

Calculating p_{remnant}. The probability that an attacker can determine that a data record is a remnant depends on the database decorrelation technique and specific application semantics.

At one extreme, an application may expose only aggregate information over data records via its queries, which prevents an attacker to locate individual data remnants or data records at all.

More realistically, however, the application may allow some queries to access the UIDs either directly or indirectly. Then if decorrelation utilizes a global placeholder, an attacker can determine with 100% certainty which records are remnants: any record with a UID equal to the global placeholder is clearly a remnant. However, if decorrelation instead relies on ghost users, 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).

An attacker can also use IIDs and knowledge of the distribution of real user profiles to calculate the probability that a record is a remnant, which would be necessary in the case in which queries cannot access UIDs directly or indirectly. Similar to how an attacker could use external knowledge about identifiers, an attacker could use knowledge about ghost profiles or ghosted records to distinguish them from real users. For example, decorrelation may generate ghost profile usernames that are random numbers or arbitrary animals, while real users may have more human-friendly usernames.

Decreasing p_{remnant}. Because an attacker may use the distribution of data records in the system to pinpoint outliers and anomalies that may be remnants, remnants may be less easily spotted if more users unsubscribe and the system contains more remnants.

Given the possibility for application-specific data to leak information even when identifiers are hidden and when ghost users are used in place of a global placeholder, the decorrelation should ensure that information exposed by remnants cannot distinguish remnants from real data records. Creating remnants from data records should not follow a clear pattern (such as assigning global values for usernames). This is highly specific to the application: it may be difficult to convincingly create user profiles for applications like Facebook and eCommerce sites, but easy in applications such as Reddit where many users use pseudonyms and have simple usage patterns.

Calculating p_{linked}. The probability that two remnants belong to the same individual can be determined by how much identifying information (from UIDs and IIDs) that attacker queries can reveal. For example, if an attacker query reveals that multiple (likely) remnants are comments on classical music, these remnants are more likely to be correlated with the

same individual than two remnants commenting on unrelated topics. If the attacker additionally queries for the time of post of these comments, and sees that both these posts were posted during daytime in California, the probability that they belong to the same individual increases (since the number of people who live in California and like classical music is less than the number of people who simply like classical music).

Decreasing p_{linked}. As with p_{remnant}, increasing the number of unsubscribed users and resulting number of remnants decreases the probability that any two remnants can be linked to any one user. For example, if there are two records that are likely to be remnants and both are upvotes on topics related to classical music, the probability that these remnants belong to the same unsubscribed user is high; but if there are thousands of likely remnants and all are upvotes on topics related to classical music, it is less likely that an attacker can tell that any two of these remnants belong to any one unsubscribed user. [lyt: I'm a bit unconvinced of this argument, but I feel like there is some intuition here].

Decreasing p_{linked} requires minimizing the amount of identifying information leaked through queries. However, there is a fundamental tradeoff between preserving useful information for the application and obfuscating remnant data.

At one extreme, if decorrelation does not modify UIDs at all (or does so in a predictable manner) and queries may return UIDs, then an attacker can learn usernames or email addresses that will identify the user. Similarly, if IIDs are not properly modified, attackers may learn information such as the location of posts or time of posts, leading to a high probability that the user can be identified. Note that this leads to a low p_{remnant} (since the remnant would look unmodified from any other subscribed user record), but also a high p_{linked}.

At the other extreme, if decorrelation modifies the remnant completely by generating random and potentially meaningless values for all remnant UIDs and IIDs, then an attacker gains little identifying information from the remnant. However, this may increase p_{remnant} significantly. Alternatively, decorrelation could simply to remove remnants completely. In either case, decorrelation limits the usefulness of remnant data to the application.

Practically, in order to both preserve the usefulness of remnants to the application and provide decorrelation, DeCor will need to balance destroying remnant information with revealing identifying information to the attacker when creating ghosts during unsubscription.

Converting UIDs to GIDs, and adding noise to IIDs. [lyt: ML/GANs seem potentially relevant here in generating fake users]

DeCor generates random gid_{key}s for every unique data record belonging to one uid_{key}, and ensure that uid_{key}s are randomly generated in the schema so that gid_{key}s are uid_{key}s are indistinguishable.

For all other UIDs, DeCor's decorrelation does the following to generate corresponding GIDs:

- Phone number: randomly selects a real area code and randomly generates a 7-digit number [lyt: take into account distribution of phone numbers in the system]
- Email: randomly selects a real area code and randomly generates an amount to add to the 7-digit number, where the RNG is seeded with the gid_{key}
- Username: generates a random but human-readable string (consisting of the concatenation of English words), using a hash of the original username and gid_{key} to select words in the username.

In order to add noise to IIDs, DeCor looks at the type of each IID column and adds noise accordingly:

- Numerical Values and Dates/Times: adds a random amount equal to the [lyt: reversible?] hash of the UID time and gid_{key} of the data record.
- [lyt: TODO add more here]

The application programmer can add GID-generating functions for other application-specific UID or IID columns via schema annotations; the programmer can also add functions to override DeCor's defaults. These annotation should generate GIDs in a manner that minimizes both p_{remnant} and p_{linked}.

[lyt: TODO: Think about how this noising can be reversed upon resubscription / use something like trapdoor permuatations? If the gid_{key}s are exposed, this might mean that an attacker could reverse the decorrelation as well... Perhaps the original uid_{key} needs to come into play.]

1.5 A Simple Web Application

To provide a concrete example of how DeCor performs decorrelation, and how p_{linked} and p_{remnant} may be calculated to determine Q and ε, we look to a simple version of a typical social media application. In this application, there are users, stories posted by users, and votes on stories with the schema shown in Figure 1. The application allows for story feed queries (selecting all of a story's content) and user profile queries (selecting all of a users' content), shown in Figure 2.

If DeCor performs zero modifications to data records when a user unsubscribes, $p_{\text{remnant}} = 0$ and $p_{\text{linked}} = 1$ after even a single user profile query. Queries reveal all stories and votes that belong to a particular user. In other words, DeCor achieves $(Q \geq 1, 1)$-decorrelation.

Now suppose DeCor uses a global placeholder strategy. DeCor zeroes u64 datatypes and datetimes, and replaces varchar values by 'deleted'). If DeCor modifies all UIDs but does not modify IIDs, then $p_{\text{remnant}} = 1$ after even a single user profile or story query. An attacker needs to perform at least two queries to identify two distinct remnant stories or votes (now

Users Table:

- *ID (uid_{key}): u64
- *Username: varchar(255)
- *Email: varchar(255)
- *Phone: varchar(255)

Stories Table:

- ID: u64
- *UserID (uid_{key}): u64
- Content: text
- **Category: u64
- **Timestamp: datetime
- **Location: decimal(18,12)

Votes Table:

- ID: u64
- *UserID (uid_{key}): u64
- **StoryID: u64
- **Timestamp: datetime

Figure 1: * indicates a UID column and ** indicates an IID column that can leak identifying information either directly or indirectly. Arbitrary user-generated content is out of scope.

Select story content:

```
SELECT users.username,
    stories.content, stories.category,
    stories.timestamp, stories.location,
    COUNT(DISTINCT votes.id) as `count`
FROM stories
    JOIN votes on stories.id = votes.storyid
    JOIN users on stories.userid = users.id
WHERE stories.id = ?
ORDER BY count;
```

Select user content:

```
SELECT users.username, users.email,
    users.phone, stories.content,
    stories.category,
    stories.timestamp,
    stories.location,
    votes.storyid, votes.timestamp
FROM users
    JOIN stories ON stories.userid = users.id
    JOIN votes ON votes.userid = users.id
WHERE users.id=?;
```

Figure 2: Queries supported by the application

associated with users with the same zeroed or 'deleted' UIDs). p_{linked} is a conditional probability: given the current distribution of users' content, what is the likelihood that two stories or votes belong to the same user? [lyt: TODO—not exactly sure how to precisely define this, even with a toy example] Thus, DeCor achieves $(Q \geq 2, p_{\text{linked}})$-decorrelation when modifying only UIDs with a global placeholder strategy.

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?)]

3 Design

DeCor relies on coarse-grained schema annotations to establish which associations to decorrelate, and builds a dataflow computation resulting in materialized views that answer application queries. Use of dataflow automatically propagates the correct updates to materialized views.

DeCor resubscribes users by transparently propagating updates to materialized views to expose real user identifiers in place of ghost identifiers.

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] 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.