Privacy Heroes Need Data Disguises

Lillian Tsai
MIT CSAIL

Malte Schwarzkopf
Brown University

Eddie Kohler
Harvard University

Abstract
Providing privacy in complex, data-rich applications is hard. Deleting accounts, anonymizing an account’s contributions, and other privacy-related actions may require the traversal and transformation of interwoven state in a relational database. Finding the affected data is already nontrivial, but privacy actions must additionally balance competing requirements, such as preserving data trails for legal reasons or allowing users to change their mind. We believe a systematic shared framework for specifying and implementing privacy transformations could simplify and empower applications. Our prototype, data disguising, supports fine-grained, nuanced, and useful policies that would be cumbersome to implement manually, including reversible transformations that can compose.

CCS Concepts
• Security and privacy → Usability in security and privacy;
• Information systems → Database management system engines.

ACM Reference Format:

1 Introduction
Applications today must support a range of data modifications to support user privacy. We call these privacy transformations. Laws like the EU’s General Data Protection Regulation (GDPR) [13] and California’s Consumer Privacy Act (CCPA) [4] codify transformations corresponding to some user actions. When a user closes their account, the site must generally delete “personally identifiable” user data, and anonymize or delete other user contributions. Prudence and good practice recommend other transformations. For example, a site might scrub or anonymize its older contents to reduce the impact of a possible later breach, since the legal consequences and reputational damage of breaches can be substantial [22, 23, 31, 35, 42]. But though they are important and legally required, privacy transformations remain difficult to implement and are arguably understudied.

In this paper, we propose data disguising, a prototype framework for specifying, implementing, and reasoning about privacy transformations. Data disguising supports a broad range of privacy transformations and separates the application of privacy transformations (“data disguises”) from application code. Developers provide disguise specifications to an external disguising tool, which computes the necessary database changes and applies them to the application’s database backend (Figure 1). Applying a disguise transforms the application data to meet a privacy-oriented goal (e.g., deleting users’ identifiers, or decorrelating identifying object relationships) while preserving application invariants and utility. It also can integrate new components, such as vaults (secure storage locations for disguised data), to support more nuanced policies than most current applications.

A systematic, yet flexible privacy transformation framework such as data disguising could reduce developer burden and ultimately improve user privacy on the web. However, realizing it will require solving several research problems. Real applications may benefit from supporting a wide range of privacy policies that can apply to the same data. Specifying those policies is challenging, and the challenge can grow when disguises interact—for example, should a disguise
that deletes data associated with a user also delete formerly-associated data that has since been anonymized? Static analysis and other techniques may be required to explain the consequences of a disguise. Second, the performance impact of data disguises could be substantial, especially for disguises that transform many database rows on a live application database. Though these important challenges remain, our proof-of-concept data disguising tool, Edna, already demonstrates the potential of this approach.

2 Challenges of Privacy Transformations

Privacy transformations have two key challenges: the broad range of possible policy choices for transformations, and the difficulty of implementing those policies. The first challenge can be appreciated by surveying current applications’ policies around account deletion. Though many applications support deletion, some retain data for legal or necessary business purposes (e.g., Spotify fraud detection [40], Amazon orders [17]); some delete public contributions, but keep private messages unanonymized and visible to their recipients, reflecting the shared nature of such messages [18, 21]; and yet others keep public contributions visible but anonymize them, reattributing the contribution to a placeholder user (e.g., GitHub’s “@ghost” [19], Reddit/Lobsters’ “[deleted]” [20, 28]). A system aiming to simplify privacy transformations must thus support a range of operations, and implement application-defined policy.

The second challenge arises because privacy transformations are inherently difficult (they require extensive tracing of user identities through application data schemas), but are also secondary to normal application functionality. Modifying or deleting data must not compromise application functionality and preserve, for example, referential integrity of an application’s relational database and other application invariants. Furthermore, one application may support several interacting privacy transformations.

The complexity of implementing basic transformations may have discouraged developers from supporting more nuanced policies. This is too bad, because where privacy is concerned, nuANCE often benefits users. For example, consider these useful policies that few applications support:

Reversible account deletion. GDPR and related laws focus on irrevocable account deletion, which permanently deletes a user’s information when they depart a platform. Users may be more likely to proactively protect their privacy if they can return, i.e., if they can reverse their account deletion. Facilitating easy return is also in the web service’s commercial interest. A weak form of reversible transformation might preserve user data in the application’s database; though this hides the data from external view, it leaves it at risk to breaches and does not satisfy most legal requirements (e.g., GDPR). Stronger forms are possible, however:

the records required to reverse account deletion might be in offline storage, or even encrypted and passed to the user themselves or to their authorized third-party cloud storage.

Expiration. Inactive users’ accounts and data can make a data breach much worse. Data expiration policies could proactively anonymize or sanitize user contributions for long-inactive users. Expiration policies should likely be reversible to support user return.

Data decay. Gradual loss of fidelity in old data might strike a balance between the need to preserve historic information and its inherent dangers. Gradual data decay policies could apply increasingly strict privacy transformations over time, aging out sensitive but outdated user data.

These policies, and especially reversible transformations, were central to our thinking as we developed the data disguise paradigm. But they come with serious technical challenges. Once applications support multiple, potentially reversible privacy transformations—which may be triggered explicitly by users (e.g., account deletion), or happen automatically and apply across users (e.g., data decay)—applications must correctly handle different interleaving of transformations that touch the same data.

3 A Nuanced Policy

To make this more concrete, we describe user scrubbing, a specific nuanced privacy transformation that the HotCRP paper review application should support. When a user deletes their account in HotCRP today, the HotCRP code transitively deletes all of the user’s data, including their reviews. However, the scientific review process generally requires that the text of reviews be retained for some time (for reference by authors and to justify decisions). Preserving reviews while deleting their owner requires a nuanced policy that combines anonymization with deletion.

Specifically, user scrubbing for a user Bea should: (1) Delete Bea’s user account. (2) Delete information that’s only relevant to Bea, such as Bea’s review preferences. (3) Delete Bea’s contact author relationships to any submissions. (4) Select or create a set of placeholder users. (5) Modify Bea’s other retained data, such as reviews, to refer to the placeholder users instead of Bea, thereby decorrelating the reviews from Bea’s identity.

After the scrubbing completes, Bea’s review texts are still in the system, but they are linked to different anonymous placeholders, making them difficult to reassociate with one another or with Bea. Placeholder users have suitable default values; for example, placeholder users should be disabled, ensuring they have no permissions and cannot log in. This policy removes Bea’s relationship to submissions, but does not remove the submissions themselves; a different policy might go even further and automatically delete a submission whose last author is scrubbed.
We propose \textit{data disguising}, a systematic approach to privacy transformations that separates them from application code. We believe that data disguising systems will help support flexible privacy transformations. Data disguising represents privacy transformations as structured disguises that the application developer specifies to capture an application’s privacy policies. Applications invoke an external data disguising tool’s API to apply disguises; the tool interprets the specification and applies the necessary physical changes to the database. Data disguising offers mechanisms to handle interactions between disguises and reversible disguises.

\subsection{Disguises: Structured Privacy Transformations}

The broad range and application-specific nature of privacy transformations poses a real implementation challenge. Most importantly, transformations must maintain the integrity of the application’s data in storage when applied. For example, decorrelating users from their reviews could easily violate referential integrity (i.e., no dangling foreign keys) unless applied carefully. Data disguises’ structured nature and automated application seeks to ensure this property.

Data disguises are built on three fundamental transformation operations—data removal, object content modification, and decorrelation by modifying references between objects—that can capture and structure many desirable privacy transformations. The application developer writes a disguise specification for each of the application’s privacy transformations. This specification consists of predicated transformation operations on each table, which describe how to transform objects of that table that satisfy the predicate. The data disguising tool takes the disguise specification and turns it into storage operations that appropriately rewrite affected foreign keys.

Figure 2 illustrates how the tool performs the example user account deletion disguise for Bea, when given a specification like that in Figure 3. When her account is active, Bea’s profile is associated with her true identity and her reviews. When Bea deletes her account, her reviews move to privacy-preserving placeholders, making it seem as if a different user entered each of Beas reviews. This prevents an observer from correlating these contributions to expose Beas identity, but importantly preserves referential integrity.

\subsection{Handling Disguise Interactions}

Applications use disguises to achieve specific privacy goals, but this can be complicated by interactions between different disguises. Because disguises inherently reduce identifying information, applying one disguise may change the outcome of future disguises applied on top of it.

For example, consider two desirable disguises in HotCRP: GDPR and ConfAnon. GDPR removes a user’s account (§3); ConfAnon provides user privacy by anonymizing all conference data. These disguises touch the same data: applying ConfAnon destroys information that GDPR would remove or transform if applied to an unmodified database.

In some cases, the disguises compose naturally—e.g., there is no need to decorrelate data that another disguise removed—but in others, the tool may need to access the original data to meet the application’s privacy goals. Data disguising provides the infrastructure and mechanisms to reveal data to support of disguise composition; however, automatically detecting when this is needed to achieve application privacy goals remains an open challenge (§7).

\textbf{Vaults} provide the infrastructure to reveal previously disguised data when necessary. A vault is a storage location not
accessible to application queries that stores reveal functions for applied disguises. Applying these functions reveals the underlying data transformed by a disguise (possibly temporarily). The disguising tool generates the reveal functions when applying a disguise, using the specification and the disguised data.

Vaults admit various deployment models that have different security and privacy properties. These include storing vaults in offline storage, which provides a modicum of security, but makes access by the data disguising tool easy; or having vaults stored entirely by some third party or locally by the user, with an API for disguise tool access. The vault contents might be encrypted, and access might require explicit approval by the user, who holds the private key. 1 Entries in a vault could also be configured to expire after some time; making the corresponding disguises irreversible.

The vault deployment model can greatly affect the practicality of disguise reversal. For example, a reversible GDPR must store reveal functions in per-user vaults external to the application storage to be GDPR-compliant. While it is reasonable to imagine accessing a single user’s vault to reverse GDPR in this model, complete reversal of ConfAnon would need to retrieve reveal functions from all users’ vaults, an infeasible task. An alternative might be to provide multi-tier security: the first tier stores reveal functions of non-GDPR disguises in a global vault accessible to the disguising tool and application, while the second tier stores reveal functions from user-invoked disguises in external, per-user encrypted vaults. We imagine that exploring different vault designs will be an important part of data disguising research.

Reverting disguises. Reveal functions also help with explicit disguise reversal (e.g., a returning user): the application invokes the disguising tool’s API to revert a previously applied disguise. This reversal permanently reveals data, restoring it to the application database. However, other disguises may have affected the database contents in the interval between the original disguising and the explicit reveal. To ensure that any revealed data still respects other active disguises, the tool keeps a persistent log of all disguises the application applied, and re-applies disguises from the relevant log interval to the revealed data.

For example, reversal of GDPR must avoid reintroducing identifiable reviews if ConfAnon has occurred since GDPR was applied. The data disguising tool would apply the relevant ConfAnon anonymization operations to any revealed data from GDPR’s reversal before making it visible the application.

1To protect against lost keys, the vault could be threshold encrypted with a private key secret-shared [39] between the user, the web application, and a trusted third party (e.g., the EFF), so that the user can authorize the application and the third party to decrypt.

<table>
<thead>
<tr>
<th>Application Disguise</th>
<th>#Object Types</th>
<th>Schema LoC</th>
<th>Disguise LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lobsters-GDPR</td>
<td>19</td>
<td>318</td>
<td>100</td>
</tr>
<tr>
<td>HotCRP-GDPR</td>
<td>25</td>
<td>352</td>
<td>142</td>
</tr>
<tr>
<td>HotCRP-GDPR+</td>
<td>25</td>
<td>352</td>
<td>255</td>
</tr>
<tr>
<td>HotCRP-ConfAnon</td>
<td>25</td>
<td>352</td>
<td>232</td>
</tr>
</tbody>
</table>

Figure 4: Data disguise specifications for Lobsters and HotCRP have similar complexity to a relational schema.

Explicit application modifications to disguised data (other than deletion) are harder to handle; the framework might prohibit them, or log them to the relevant vaults.

5 Prototype

Our prototype disguising tool, Edna, is written in Rust and provides data disguising for applications that use relational databases. Disguises in Edna associate each table in the application schema with a set of predicate-transformation pairs. Predicates are arbitrary SQL WHERE clauses, which Edna uses to select table rows to transform; a transformation is either a removal, a decorrelation of a particular foreign key, or a modification of a particular column. A modification takes a closure over the original column value that returns the updated value. For tables that are decorrelated, developers describe how to find placeholders by providing per-column closures over the original column value that return the placeholder column value.

Edna represents vaults as (currently unencrypted) per-user database tables. Reveal functions stored in vaults use the original and updated states of objects touched by a reversible disguise to generate the necessary operations to restore the original state. Edna also keeps a disguise history table that logs all disguises performed.

6 Case Studies

To evaluate the ease of writing disguises, we implement disguises for GDPR deletion in Lobsters [27], an open-source feed application, and HotCRP [16]. We consider four disguises: Lobsters-GDPR and HotCRP-GDPR implement the current account deletion policies in the two applications [24, 28]. HotCRP-GDPR+ specifies a HotCRP account deletion policy that balances useful data retention with data deletion for privacy (user scrubbing, §3), and HotCRP-ConfAnon specifies the conference anonymization disguise for HotCRP.

Developer Effort. We would hope that writing disguises involves similar labor and difficulty as writing relational schemas. In particular, a developer should write a disguise only once, and specify some reasonable number of predicated transformations for each object type. Figure 4 shows that the disguise specification for our applications is indeed comparable in size to the applications’ schemas.
Performance. Data disguising faces several performance challenges. As expected, we observe that the number of queries performed by Edna to fetch and update the relevant to-be-disguised objects grows linearly with the number of objects. This disguise overhead is unavoidable: these modifications are crucial to applying the disguise. Edna currently applies these changes in one large SQL transaction; batching, parallelization, and asynchronous application could improve performance. The importance of reducing the cost of disguise application depends on the rate of disguising, which may range from rare (as in today’s applications) to quite frequent (in a privacy-supporting world where users freely disguise and reveal themselves, or data expires).

Disguise interdependencies can further increase the cost of disguising: when applying a disguise, Edna not only modifies objects, but may also read and apply reveal functions from vaults. In the worst case, Edna might need to read, reverse, and reapply all previous reversible disguises in their entirety. We evaluate the estimated cost of vault operations and disguise composition in an experiment with a HotCRP database with 430 users (30 PC members), 450 papers, and 1400 reviews. We first invoke Edna with two independent disguises: two HotCRP-GDPR+ disguises for different users. We then compare the cost of invoking HotCRP-GDPR+ after having applied HotCRP-ConfAnon, a conflicting but reversible disguise. Applying a PC member’s HotCRP-GDPR+ after having applied an independent HotCRP-GDPR+ takes 135ms on average. The same HotCRP-GDPR+ disguise applied after HotCRP-ConfAnon takes 452ms on average; HotCRP-ConfAnon itself takes about 7,000ms. The added overhead stems from Edna’s temporarily recorrelation of objects using reveal functions so it can correctly remove them. While this selective reintroduction of data from user vaults is not free, it is cheaper than completely undoing the prior HotCRP-ConfAnon disguise.

When we apply a (manual) optimization that avoids unnecessarily redoing recorrelation actions that have already been taken by HotCRP-ConfAnon, the latency for HotCRP-GDPR+ after HotCRP-ConfAnon drops to 118ms. We imagine that we will be able to use static analysis of the disguise and schema to automate this optimization in the future.

7 Discussion
Data disguising simplifies implementing privacy transformations. However, our proposed data disguising framework still leaves open questions.

For instance, we provide mechanisms for a disguising tool to handle disguise composition, but do not yet answer how the disguise specification communicates an application’s privacy goals and how the tool decides when to utilize vaults and reveal functions. If only one disguise is ever applied (e.g., GDPR), the disguise specification’s operations may completely state the application’s privacy goals (e.g., de-identify the user). However, with multiple, co-dependent disguises, the tool may reach an end-state that fails to achieve the application’s privacy goals if it merely applies the disguise operations. How to convey the application’s privacy goals to the disguising tool is a challenging research problem. One possibility might be to ask developers to write assertions over the end-state of application data after a disguise has been applied (e.g., “user no longer has any reviews”). If these assertions were amenable to static analysis, the tool could, before applying any disguises, determine which mechanisms need to be utilized to appropriately apply each disguise. Or perhaps assertions could be arbitrary predicates over the end-state, which the tool would check after disguise application to ensure the state adheres to the application’s privacy goals; if these checks fail, the tool would revert the disguise and try again with a different mechanism until it passes the checks, or notify the developer of an error.

Furthermore, our framework does not answer how disguises compose with normal application changes to disguised data. Application updates may alter the scope of a disguise (i.e., changing the set of objects satisfying a disguise predicate). In addition, data introduced by a reveal function may lack application updates that occurred since the original disguise. One possible solution is to make such updates themselves disguises, and store metadata about them in vaults, but this would be expensive. Another solution would prohibit updates to disguised data (which limits the application).

Finally, more research is required to handle updates to the application schema or disguise specifications in a system that has already applied disguises. Database schema evolution research [10] may offer insights into supporting disguises and guise reversals after such updates.

Data disguising also has some clear limitations. It assumes that application objects capture all data that needs transforming; application snapshots, backups, or external data copies are out of scope, and require e.g., taint tracking techniques. Data disguising also importantly does not provide privacy guarantees beyond the specification. Privacy goals are necessarily application-specific, and data disguising is only as good as the developer-written specification. We imagine that data analysis tools and heuristics can help developers improve or catch errors in disguise specifications, similar to e.g., techniques for detecting incorrect deletion [9].

8 Related Work
Two prior systems for personal data deletion are particularly relevant to data disguising: Sypse [11], which pseudonymizes user data and partitions personally identifying information (PII) from other data, and DELF [9], a framework for data deletion at Facebook. While Sypse also traverses foreign keys, its design is application-oblivious, and leaves the
specification of what counts as PII as future work. Data disguising provides a way to specify sensitive data and correlations, as well as application-specific transformations. Instead of constantly maintaining data partitions (as Sypse does), data disguising adapts the physical database on disguising. DELF [9] helps developers write correct data deletion code, and uses annotations on application edge and object types to specify a deletion policy. DELF focuses only on data deletion, while data disguising targets broader privacy transformations, including decorrelation.

**Database designs for GDPR compliance** [25, 38] track the owner of data objects and erase them when requested under the GDPR. They either modify the data layout [38] or use fine-grained information flow tracking (IFC) to determine PII propagation and restrict access [25]. Data disguising can be employed for GDPR compliance, but supports nuanced privacy transformations beyond deleting PII, and requires no ownership tracking or fine-grained IFC.

Other systems enforce developer-specified visibility and access control policies based on general information flow control approaches [8, 14, 15, 37, 41], authorized views [3] or per-user views [29], and rewriting database queries [30, 34]. Data disguising transforms the actual data stored.

Privacy-preserving data mining approaches, such as k-anonymity, l-diversity, and differential privacy [1, 12], provide statistical privacy guarantees. These complement data disguising: disguise predicates might be based on differential privacy, for example.

Finally, clean-slate designs for user-centric data ownership paradigms seek to “decentralize” the internet [2, 5–7, 26, 32, 33, 36], granting ultimate control over data to end-users. These systems are an extreme realization of per-user vaults: they typically lack the capacity for server-side compute, burden users with long-term data maintenance, and break the current application revenue model. In contrast, data disguising helps developers specify and automate privacy transformations without changing the application data model or business model.

**Acknowledgments**

We thank M. Frans Kaashoek, Shriram Krishnamurthi and members of the MIT PDOS group for feedback on earlier version of this paper. This work was supported by NSF awards CSR-1704172, CSR-1704376, and CSR-2045170, and by a Google Research Scholar Award. Lillian Tsai is supported by an NSF GRFP.

**References**


[27] Lobsters. URL: https://lobsters.rs (visited on 02/03/2021).


