Flexible Privacy via Disguising and Revealing

by

Lillian Tsai

A.B., Harvard University (2017)
S.M., Harvard University (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

© Massachusetts Institute of Technology 2024. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
May 3, 2024

Certified by .....
M. Frans Kaashoek
Charles Piper Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Co-Certified by .....
Malte Schwarzkopf
Professor of Computer Science, Brown University
Thesis Co-Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Flexible Privacy via Disguising and Revealing

by

Lillian Tsai

Submitted to the Department of Electrical Engineering and Computer Science
on May 3, 2024, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Many users today have tens to hundreds of accounts with web services that store sensitive data, from social media to tax preparation and e-commerce sites [12, 27, 63]. And while users now have the right to delete their data (via e.g., the GDPR [26] or CCPA [11]), users want and deserve more nuanced controls over their data that don't exist today. For example, a user might wish to hide and protect data of an e-commerce or dating app profile when inactive, but also want their data to be present should they return to use the application. Today, however, services often provide only coarse-grained, blunt tools that result in all-or-nothing exposure of users' private information.

This thesis introduces the notion of *disguised data*, a reversible state of data in which sensitive data is selectively hidden. This thesis then describes Edna—the first system for disguised data—which helps web applications allow users to remove their data without permanently losing their accounts, anonymize their old data, and selectively dissociate personal data from public profiles. Edna helps developers support these features while maintaining application functionality and referential integrity via *disguising* and *revealing* transformations. Disguising selectively renders user data inaccessible via encryption, and revealing enables the user to restore their data to the application. Edna's techniques allow transformations to compose in any order, e.g., deleting a previously anonymized user's account, or restoring an account back to an anonymized state.

Thesis Supervisor: M. Frans Kaashoek
Title: Charles Piper Professor of Electrical Engineering and Computer Science

Thesis Co-Supervisor: Malte Schwarzkopf
Title: Professor of Computer Science, Brown University

Acknowledgments

Better acks

We thank Akshay Narayan, Anish Athalye, Derek Leung, Gohar Irfan Chaudhry, Henry Corrigan-Gibbs, James Mickens, Kevin Liao, Kinan Dak Albab, Matthew Lentz, Nickolai Zeldovich, and the anonymous reviewers for their comments over the years that much improved this paper. Nick Young helped implement the Lobsters case study. We also thank members of MIT's PDOS group, Brown's ETOS group, and the SystemsResearch@Google Team for discussing and providing feedback on Edna. Finally, we are grateful to Aurojit Panda, our shepherd, for his helpful comments and for suggesting multiple improvements to the system and paper.

This work was supported by NSF awards CNS-2045170 and CSR-1704376, a Google Research Scholar award, and funding from VMware. Lillian Tsai was supported by an NSF Graduate Research Fellowship. We also thank CloudLab [25] for providing resources for this paper's artifact evaluation.

Contents

List of Figures

List of TODOs

Chapter 1

Introduction

Many users today have tens to hundreds of accounts with web services that store sensitive data, from social media to tax preparation and e-commerce sites [12, 27, 63]. And while users now have the right to delete their data (via e.g., the GDPR [26] or CCPA [11]), users want and deserve more nuanced controls over their data that don't exist today. For example, a user might wish to hide and protect data of an e-commerce or dating app profile when inactive, but also want their data to be present should they return to use the application. Today, however, services often provide only coarse-grained, blunt tools that result in all-or-nothing exposure of users' private information, and fail to achieve the flexible privacy that users want and deserve.

This thesis introduces the notion of *disguised data* for flexible data privacy. Disguised data is a reversible state of data in which sensitive data is selectively hidden. To demonstrate the feasibility of disguised data, this thesis also presents Edna—the first system for disguised data—which helps web applications allow users to remove their data without permanently losing their accounts, anonymize their old data, and selectively dissociate personal data from public profiles. Edna lets developers support these features while maintaining application functionality and referential integrity via *disguising* and *revealing* transformations. Disguising selectively renders user data inaccessible via encryption, and revealing enables the user to restore their data to the application.

In this chapter, we motivate the need for data disguising for flexible privacy; describe the challenges in successfully disguising and revealing data; and introduce our approach to achieve data disguising as well as our contributions.

1.1 Motivation

User today lack desirable controls over their personal data stored by web applications. Consider Twitter: after a change in management [18], many users wanted to leave the platform and try out alternatives (e.g., Mastodon [43]). But each user faced a tricky question: should they keep their Twitter account, or should they delete it? Advice on how to quit Twitter [6, 44] highlight how keeping an inactive account leaves sensitive information (e.g., private messages)

vulnerable on Twitter's servers; but deleting the account prevents the user from changing their mind and coming back, causing them to permanently lose all their followers and content. Hence, many users left Twitter but kept their accounts [5, 48, 56]. A better solution would let users temporarily revoke Twitter's access to their data while having the option to come back.

Similarly, users give dating apps personal data, and frequently deactivate and reactivate their accounts. This sensitive data should be protected from the application and potential data breaches [22, 47] when a user deactivates their account, but be readily available when they choose to return.

Users may also prefer old data, such as past purchases in an online store or their passport details with a hotel, to be inaccessible to the service after some time of inactivity, and therefore protected from leaks or service compromises [52, 65]. Or users may prefer to—explicitly or automatically—dissociate their identity from old data, such as teenage social media posts or old reviews on HotCRP. Today, users work around the lack of such support by explicitly maintaining multiple identities (e.g., Reddit throwaway accounts [51] and Instagram "finstas" [70]), an inflexible and laborious solution.

Providing this functionality can benefit both the service and the user. It helps the service comply with privacy regulations, reduces its liability on data breaches, and appeals to privacy-conscious users; meanwhile, the user can rest assured that their privacy is protected, but can also get their data back and reveal their association with it if they want.

1.2 Disguised Data: A New Abstraction for Flexible Privacy

This desired flexible privacy functionality describes an in-between state of user data in web applications: where users are not quite gone, because they can return to the application; but they are also not quite there, because some or all of their data has been removed.

To capture this new state of data, this thesis introduces the new key abstraction of *disguised data*. Disguised data represents a state of data where (*i*) some or all of the user's original sensitive data is rendered inaccessible to the application; (*ii*) some data may be replaced with placeholders to keep the application structure intact (e.g., placeholder parent comments to maintain comment thread structure); and (*iii*) the data can be restored with the user's authorization.

Systems for disguised data move closer to an Internet where users can leave services and return at any time, where old data on servers is protected by default, and where services provide users with control over their identifying data visible to the service and other users.

1.3 Challenges

Today, disguised data for flexible privacy remains out of reach for users of web applications in part because getting it right is hard. Real applications have complex notions of privacy, data ownership, and data sharing. Simple solutions that e.g., delete all data associated with a user

can break referential integrity or create orphaned data, which requires application changes to handle correctly, and lack support for users to return. To solve this manually, a developer would have to carefully perform application-specific database changes to remove data, store any data removed to be able to later restore it, and correctly revert the database changes on restoring.

Developers would also have to reason about interactions between multiple data-redacting features, and how these features would compose. For example, imagine an application that supports both account deletion and anonymizing old data: if a user wants to delete all their posts after they have been anonymized, a SQL query must somehow determine which anonymized posts belong to the user in order to remove them. And if the user later wants to return, the developer must account for the applied anonymization and restore posts as anonymized.

Furthermore, stored removed data must be inaccessible to the application and protected against data breaches, but must be accessible if the user chooses to return. The developer also needs to provide user-friendly ways for users to prove their ownership of the stored data.

1.4 Our Approach

To realize disguised data, we present a general system that helps developers specify and apply two kinds of transformations: *disguising transformations*, which move the user's data into a disguised state; and *revealing transformations*, which restore the original data at a user's request. Disguising transformations aim to protect the confidentiality of users' disguised data (e.g., links to throwaway accounts or old HotCRP reviews) even if the application is later compromised (e.g., via a SQL injection or a compromised admin's account).

We demonstrate our approach in Edna, a system that realizes disguising and revealing transformations for database-backed web applications via a set of primitives that have well-defined semantics and compose cleanly. Developers specify the transformations that their application should provide, and Edna takes care of correctly applying, composing, and optionally reverting them, while maintaining application functionality and referential integrity.

Our approach for Edna's approach faces three challenges, and we describe each challenge and our solution's abstractions next.

First, Edna needs to present a simple, yet versatile interface for developers to specify disguising transformations. Edna addresses this challenge with a restricted programming model centered around three primitives: remove, modify, and decorrelate (which reassigns data to placeholder users). This model limits the potential for developer error, and lets Edna derive the correct disguising and revealing operations, while supporting a wide range of transformations.

Second, to work with existing applications in practice, Edna's disguising transformations should require minimal application modifications. To achieve this, Edna introduces *pseudoprincipals*, anonymous placeholder users that are inserted into the database on disguising and exist solely to own data decorrelated from real users (e.g., because the application requires the data to continue operating) and maintain referential integrity. Pseudoprincipals can also act as built-

in "throwaway accounts," as they let the user disown data after-the-fact, as well as potentially later reassociate with it. To correctly reason about ownership when data may be decorrelated multiple times (e.g., by global anonymization after throwaways have been created), Edna maintains an encrypted *speaks-for chain* of pseudoprincipals that only the original user can unlock and modify.

Third, Edna needs to have access to the original data for users to be able to reveal their data and return to the application, but the whole point is to make that data inaccessible to the service. While Edna could ask users to store their own disguised data, this would be burdensome. Instead, Edna stores the disguised data on the server in encrypted form as *diff records*, and unlocks and restores data to the service only when a user provides their *reveal credentials* (e.g., a password or a private key).

1.5 Contributions

The main contribution of this thesis is the identification and exploration of *disguised data* for flexible user data privacy in web applications. As part of this, this thesis contributes:

1. The abstraction of *data disguising via disguising and revealing transformations*, including disguise specifications written with a small, composable set of data-anonymizing primitives (remove, modify, and decorrelate to pseudoprincipals), and reveal credentials to allow users to reveal their data (Ch. 3).
2. Design techniques to realize disguised data, including diff records and speaks-for chains. (Ch. 4)
3. Edna, a prototype Rust library for MySQL-backed web applications that implements user data control via disguising and revealing, which is open-source at `https://github.com/tslilyai/edna` (Ch. 5).
4. Case studies that integrate Edna with three real-world web applications and demonstrate Edna's ability to enable composable and reversible transformations. (Ch. 6)
5. An evaluation of Edna's effectiveness and performance, including how Edna contrasts with and complements related work (Qapla [45] and CryptDB [53]) (Ch. 7)

While disguised data can help developers to add more flexible user data controls, the abstraction and its implementation in Edna have some limitations. First, disguised data is a concept scoped for single applications, and does not tackle the problem of data sharing between services. Edna also assumes bug-free disguise specifications, and that applications use Edna correctly. Furthermore, Edna does not aim to protect undisguised data in the database against compromise; combining Edna with an encrypted database can add this protection. Finally, attacks to identify users from Edna's metadata (e.g., the size of stored disguised data) or placeholder data left in the database (e.g., embedded text) are out of scope.

Chapter 2

Related Work

Edna addresses the problem of disguised data for flexible data privacy in web applications. Existing systems aim instead to support data deletion, prevent unauthorized data access, or protect against database server compromise—valuable, but complementary goals to achieving disguised data.

2.1 Compliance Tools

Compliance tools such as K9db [19] and GDPRizer [2] help correctly extract or delete data in correspondence to compliance-related requests. These tools include systems to support wholesale user data extraction or deletion by tracking data ownership by modifying the data layout [19, 61], tracking information flow [40], or changing the storage hardware [35].

Tools like DELF at Meta [17] focus on the related problem of data deletion, letting developers specify deletion policies via annotations on social graph edges and object types. DELF ensures correct cascading data deletion, both for compliance-related data deletion requests, and for normal application deletions.

While disguising data can provide GDPR-compliant account deletion, disguising also supports more nuanced use cases beyond simple deletion: for example, Edna allows users to return after deletion, hides old data for inactive users, or hides some but not all data so the user can continue using the application. Edna could, however, benefit from these systems' proposed techniques to track a user's data.

2.2 Policy Enforcement Systems

Policy enforcement systems aim to prevent unauthorized access to data and protect against leakage via compromised accounts or SQL injections. They enforce developer-specified visibility and access control policies via information flow control [16, 29, 33, 59, 72], authorized views [9], per-user views [42], or by blocking or rewriting database queries [45, 50, 74].

Policy-enforcing systems do not help users anonymize data or maintain application integrity constraints, which is the explicit goal with disguised data. Database contents of sensitive data—the data under disguise—are also modified so the data is no longer available in the database, and thus unavailable even to the service itself. This is unlike policy enforcement systems, which can deny access to sensitive data, but will still retain it in the database.

2.3 Encrypted Storage Systems

Encrypted storage systems such as CryptDB [53] and Mylar [54] protect against database server compromise, with some limitations [31]. These systems encrypt data in the database, and ensure that only users with access to the right keys can decrypt the data. Applications must handle keys, and send queries either through trusted proxies that decrypt data [53], or move application functionality client-side [54]. Encrypted databases have orthogonal goals to systems for disguised data: while they protect data at all times against attackers who do not have the keys, encrypted databases do not help applications anonymize or temporarily remove data, which systems for disguised data do. Any user with legitimate access can view the data in an encrypted database, whereas disguised data is removed from the database.

2.4 Other Related Work

Other work has also focused on protecting user data and giving users more control over their data's exposure. However, these systems differ from Edna in the specific problem they aim to solve, the threat model against which they protect, and how they can be deployed with today's applications.

Some platforms focus on enforcing user-defined policies, instead of developer-defined policies like Edna. They prove that server-side processing respects user-defined data policies via cryptographic means [10] or systems security mechanisms [69]. However, this may restrict feasible application functionality (e.g., to additively homomorphic functions), or restrict combining data with different policies, and requires extensive modifications to existing applications to redeploy.

Systems like Pesos [39] and Ironsafe [67] use trusted execution environments (TEEs) to enforce policy compliance in third-party storage platforms without trusting them with user data. Ryoan [34] also runs data processing applications as sandboxed modules within TEEs, thus protecting user data against untrusted applications in addition to untrusted platforms. These systems rely on the security of TEEs, which allows them to protect against their stronger threat models. However, this requires applications to be written with the systems' specific architecture in mind, making these systems more difficult to deploy in today's applications. By contrast, Edna trusts developers to be well-intentioned, and undisguised data can be accessed by the application.

Decentralized platforms such as Solid [58], BSTORE [15], Databox [46], and others [3, 13, 14, 41, 49] put data directly under user control, since users store their own data. But decentralized platforms burden users with maintaining infrastructure, lack the capacity for server-side compute, and break today's ad-based business model. By contrast, Edna leaves the data and business models unchanged, and stores all data, including disguised data, on the application's servers.

Devices using iOS [4], Android [4], or CleanOS [64] revoke data access via encryption, like Edna does. However, these systems operate in settings that store only a single user's data; disguised data applies in settings that include multiple users' data and shared data.

Vanish [28] provides users with self-destructing data and a proof of data deletion using decentralized infrastructure and cryptographic techniques (with limitations against Sybil attacks [71]). Unlike Edna, Vanish cannot restore deleted data and requires extensive application restructuring to deploy.

Sypse [23] pseudonymizes user data and partitions personally identifying information (PII) from other data. Instead of partitioning data, Edna modifies the database and stores disguised data encrypted.

Finally, oblivious object stores like Dory [20] and Snoopy [21] protect data and search access patterns against powerful adversaries who can e.g., compromise the entire cloud software stack and view metadata like network traffic and access patterns. To provide strong security, these systems rely on complex encryption schemes, oblivious RAM, and other cryptographic techniques; however, these techniques more greatly affect performance and deployability compared to Edna.

Chapter 3

Overview

This section introduces the concept of disguised data by illustrating how developers and users would interact with an application that supports data disguising and data revealing.

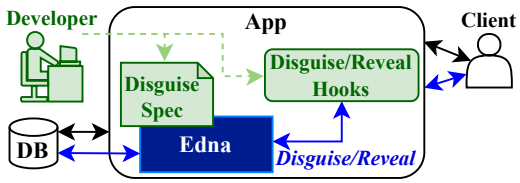


Figure 3-1: Developers write disguise specifications and add hooks to invoke Edna from the application (green); in normal operation, clients use these hooks in the application to disguise and reveal their data in the database (blue).

Edna helps developers realize new options for users to control their data via *disguising transformations*; and add support for users to invoke *revealing transformations* The developer integrates an application with Edna by writing disguise specifications and adding hooks to disguise or reveal data using Edna's API (Figure 3-1). This proceeds as follows:

(1) An application registers users with a public–private keypair that either the application or the user's client generates; Edna stores the public key in its database, while the user retains the private key for use in future reveal operations.

(2) When the application wants to disguise some data, it invokes Edna with the corresponding developer-provided disguise specification and any necessary parameters (such as a user ID). Disguise specifications can remove data, modify data (replacing some or all of its contents with placeholder values), or decorrelate data, replacing links to users with links to pseudoprincipals (fake users). Edna takes the data it removed or replaced and the connections between the user and any pseudoprincipals it created, encrypts that data with the user's public key, and stores the resulting ciphertext—the *disguised data*—such that it cannot be linked back to the user without the user's private key.

check (3) If the developer decides to update application contents (e.g., by performin a global moderation or a schema migration), then the developer writes an *application update executable*

```
// Decorrelate comments on stories w/tag {{TAG}}
"comments": [{
  "type": "Decorrelate",
  "predicate": "tags.tag = {{TAG}}",
  "from": "comments JOIN stories ON comments.story_id = stories.id
           JOIN taggings ON stories.id = taggings.story_id
           JOIN tags ON...",
  "group_by": "stories.id",
  "principal_fk": "comments.user_id" } ],
// Remove votes on stories w/tag {{TAG}}
"votes": [{
  "type": "Remove",
  "predicate": "tags.tag = {{TAG}}",
  "from": "votes JOIN stories...",
  "principal_fk": "votes.user_id",
}, ... ]
```

Figure 3-2: Lobsters topic-based anonymization disguise specification (JSON pseudocode), which decorrelates comments and removes votes on stories with the specific topic tag.

that encapsulate these changes applied to a single user's data, and invokes Edna's API to record this update. Edna then uses these executables to reveal disguised data without overwriting the associated application changes.

(4) When a user wishes to reveal their disguised data, they pass credentials to the application, which calls into Edna to reveal the data. Credentials are application-specific: users may either provide their private key or other credentials sufficient for Edna to re-derive the private key. Edna reads the disguised data and decrypts it, undoing the changes to the application database that disguising introduced.

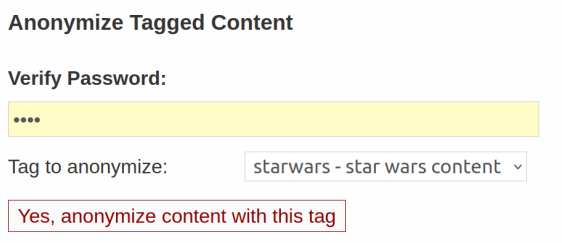
Edna provides the developer with sensible default disguising and revealing semantics (e.g., revealing makes sure not to overwrite changes made since disguising).

3.1 Example: Lobsters Topic Anonymization

We next describe how Edna's API and disguise specifications work via a disguising transformation for Lobsters [1]. Lobsters [1] is a link-sharing and discussion platform with 15.4k users. Its database schema consists of stories, tags on stories, comments, votes, private messages, user accounts, and other user-associated metadata. Users create accounts, submit URLs as stories, and interact with other users and their posted stories via comment threads and votes.

Consider **topic-based anonymization**, which allows users to hide their interest in a topic (a "tag" in Lobsters) by decorrelating their comments and removing their votes on stories with that tag. For instance, a Lobsters user Bea who posts about their interests—Rust, static analysis, and Star Wars—might want to hide associations with Star Wars before sharing their profile with potential employers. This is currently not possible in Lobsters.

The Lobsters developer can realize topic-based anonymization as a disguising transformation. First, the developer writes a disguise specification that instructs Edna to decorrelate com-




```
def tag_anon
  # instantiate the disguise spec with the provided tag to anonymize
  disg_spec = edna.instantiate_spec("tag_anon.json",params[:tag])
  # apply the disguising transformation
  disg_id = edna.apply_disguise(@user.id,params[:passwd],disg_spec)
  # email the disguise ID to the user to allow revealing
  SendDisguiseEmail(@user, disg_id)
end
```

Figure 3-3: The Lobsters developer adds a hook in the UI and code to perform topic-based anonymization.

ments and remove votes on "Star Wars" stories (Figure 3-4), and provides it to Edna. They also add frontend code and UI elements that allow authenticated users to trigger the disguising transformation (Figure 3-3). When Bea wants to anonymize their contributions on content tagged "Star Wars," Lobsters invokes Edna with the provided specification.

xxx If the developer performs a global application change that e.g., normalizes the URLs of stories, the developer performs a bulk update in the Lobsters applications. The developer also writes an update executable encoding how to update individual user's stories' URLs as well. Then the developer invokes Edna's `record_update` API call with the update executable.

When Bea wants to deanonymize their "Star Wars" contributions, Lobsters invokes Edna with Bea's reveal credentials and the disguise ID, and Edna reveals Bea's associations with "Star Wars" posts, applying application updates and schema migrations while doing so.

3.2 Disguise Specifications

Edna's developer interface is designed so that developers need to reason only about the high-level semantics of disguising expressed in the disguise specification. Edna takes care of applying disguise transformations, composing it with other disguising and revealing transformations, and creating and storing disguised data.

Disguise specifications tell Edna what application data objects to disguise and how to disguise them. A disguise specification identifies objects by database table name, principal, and predicate, where a predicate is a SQL `WHERE` clause. Edna by default disguises all objects related to the given principal, as defined by a foreign-key relationship provided in the disguise specification (using `principal_fk`), but predicates can narrow the transformation's scope (e.g., to stories with specific tags). For each selected group of objects, developers choose to *remove*, *modify*, or *decorrelate* them. The example specification in Figure 3-4 decorrelates all comments

```
// Decorrelate comments on stories w/tag {{TAG}}
"comments": [{
  "type": "Decorrelate",
  "predicate": "tags.tag = {{TAG}}",
  "from": "comments JOIN stories
           ON comments.story_id = stories.id
           JOIN taggings
           ON stories.id = taggings.story_id
           JOIN tags ON...",
  "group_by": "stories.id",
  "principal_fk": "comments.user_id" } ],
// Remove votes on stories w/tag {{TAG}}
"votes": [{
  "type": "Remove",
  "predicate": "tags.tag = {{TAG}}",
  "from": "votes JOIN stories...",
  "principal_fk": "votes.user_id",
}, ... ]
```

Figure 3-4: Lobsters topic-based anonymization disguise specification (JSON pseudocode), which decorrelates comments and removes votes on stories with the specific topic tag.

and removes all votes on stories with a particular tag, specified by the `TAG` parameter provided at invocation time.

does this flow? Modification and decorrelation require Edna to generate placeholder content for modified columns and/or new pseudoprincipals. Developers use one of Edna's simple generation policies to tell Edna how to generate per-column placeholder values. Edna can generate:

- a specified constant number,
- a specified constant string,
- a random number between some lower and upper bounds,
- a random string of specified length,
- a random email address,
- a random phone number,
- a random timestamp,
- a specified constant boolean, and
- NULL.

To ensure that decorrelation preserves referential integrity, Edna generates pseudoprincipals to replace the original principal. Decorrelation can use pseudoprincipals at different granularities. In the extreme, the disguise specification may tell Edna to create a unique pseudoprincipal for each decorrelated application object. In our example, however, all comments by the same user on the same story decorrelate to the same pseudoprincipal (`"group_by": "stories.id"`), thus keeping same-story comment threads intact. The same user's comments on different stories, however, decorrelate to different pseudoprincipals; an alternative might group comments by `comments.user_id`, so a single pseudoprincipal adopts all of a user's `TAG`-related comments (effectively creating a "Star Wars" throwaway account).

Developers can inform Edna to, upon revealing, check for any objects added after disguising that refer to pseudoprincipals; for example, a decorrelated comment might have new responses. This enables Edna to preserve referential integrity for data referring to pseudoprincipals. Edna provides three options if it finds such objects: (*i*) change the object's reference to point to the original principal; (*ii*) delete the object; and (*iii*) continue referring to the pseudoprincipal. This option is global; another design for Edna could also support a menu of options, such as per-table checks and fixes (where the developer to specifies per-table policies) or per-inserted-object ones (where the developer makes application modifications to log all added references to pseudoprincipals).

3.3 Application Update Executables

Developers provide Edna with information about application updates or schema migrations to allow Edna to reveal data without overwriting or conflicting with these changes. When the developer performs operations that need to hold over revealed data—e.g., moderations—they will additionally invoke Edna's API, allowing Edna to record and appropriately apply application updates.

Edna expects developers to provide application update executables that takes a user's disguised data rows and produces updated-disguised data rows. Oftentimes, these executables may be a subroutine of the bulk update that performs the whole application change. For example, a developer may want to normalize all URLs in `posts`; they would (*i*) read all posts, (*ii*) normalize the URL in each post individually, and then (*iii*) `INSERT...UPDATE` the normalized posts. An application update executable would simply capture step (*ii*), producing normalized URL posts for the posts in a user's disguised data.

However, the bulk update may use queries that apply to entire database tables instead of rows. In these scenarios, the developer should write a separate update executable for updating disguised data rows. For example, a developer may want to add a column to the `posts` table. The bulk update simply performs an `ALTER TABLE` query, while the update executable receives post rows without the column as input, and outputs post rows with the column.

3.4 User Reveal Credentials

Edna provides developers and users with the abstraction of *reveal credentials*, which users provide to the application to authorize revealing their data. Reveal credentials allow applications that use Edna to support familiar user authentication workflows when users want to reveal their disguised data.

Edna supports two forms of reveal credentials: (*i*) the principal's private key itself; or (*ii*) the principal's application password or a recovery token (in case they forget their password), either of which Edna can use to rederive the private key. Developers can use either or both of

these credentials depending on application needs. In our Lobsters example, Edna rederives the user's private key using their password. Password or keypair changes require an application to re-register the user with Edna, which generates new recovery tokens and re-encrypts the user's disguised data.

3.5 Threat Model

Developers and users can reason about the security properties of disguised data given Edna's threat model. Edna protects the confidentiality of disguised data between the time when a user disguises their data and the time when they reveal it. During this period, Edna ensures that the application cannot learn the contents of disguised data, nor learn what disguised data corresponds to which user, even if the application is compromised and an attacker dumps the database contents (e.g., via SQL injection). Edna stores disguised data encryptedly, so its confidentiality stems from "crypto shredding," a GDPR-compliant data deletion approach based on the fact that ciphertexts are indistinguishable from garbage data if the key material is unavailable [24, 30, 55, 66].

We make standard assumptions about the security of cryptographic primitives: attackers cannot break encryption, and keys stored with clients are safe. If a compromised application obtains a user's credentials, either because the user provides them to the application for reveal, or via external means such as phishing, Edna provides no guarantees about the user's current or future disguised data. Edna also expects the application to protect backups created prior to disguising, and external copies of the data (e.g., Internet Archive or screenshots) are out of scope.

While Edna hides the contents of disguised data and relationships between disguised data and users, it does not hide the existence of disguised data. (An attacker can see if a user has disguised some data, but cannot see which disguised data corresponds to this user.) An attacker can also see any data left in the database, such as pseudoprincipal data or embedded text. Edna puts out of scope attacks that leverage this leftover data and metadata to infer which principal originally owned which objects.

Edna's choice of threat model and its limitations stem from Edna's goal of practicality and usability by existing applications, and from design components that support this goal. For example, decorrelation with pseudoprincipals removes explicit user-content links, but leaves placeholder information in the database to avoid application code having to handle dangling references. Similarly, leveraging server-side storage to hold disguised data leaves metadata available to attackers, but avoids burdening users with data storage management.

Chapter 4

Edna's Design

This chapter describes the design of Edna and the techniques Edna uses to support disguising and revealing. We first look at how Edna performs disguising transformations, and then dive into the details of how Edna reveals disguised data. We then describe the more complicated scenarios possible with Edna, namely disguise composition, shared data, and action as pseudoprincipals. This chapter concludes with an analysis of the security of Edna's design with respect to the threat model described in §3.5.

4.1 Disguising

To apply a disguising transformation, Edna creates a unique *disguise ID* and queries for the data to disguise based on the disguise specification predicates. To preserve referential integrity, Edna constructs a dependency graph between tables based on foreign key relations (assuming no circularity), and first performs removes from tables in topologically-sorted order. Edna then performs decorrelations and modifications in specification order, potentially generating and storing pseudoprincipals.

Next, to record disguised data, Edna generates *diff records* that contain (*i*) the original data row(s), and (*ii*) placeholder data rows the disguise inserted or rewote in the application (e.g., pseudoprincipals or the value of any modified columns). All types of diff records also contain the disguise ID. The original and placeholder rows contained by a diff record varies by disguise operation as follows:

- Remove diff records contain (*i*) the removed original row, and (*ii*) no placeholder rows;
- Modify diff records contain (*i*) the unmodified row, and (*ii*) the row with the modified value; and
- Decorrelate diff records contain (*i*) the referencing row with the original foreign key value, and (*ii*) the referencing row with the placeholder foreign key value pointing to a pseudoprincipal, and the pseudoprincipal's row.

For each new pseudoprincipal, Edna generates a public–private keypair and adds a *speaks-for record*, which adds a link to the original principal's *speaks-for chain*. A speaks-for record contains

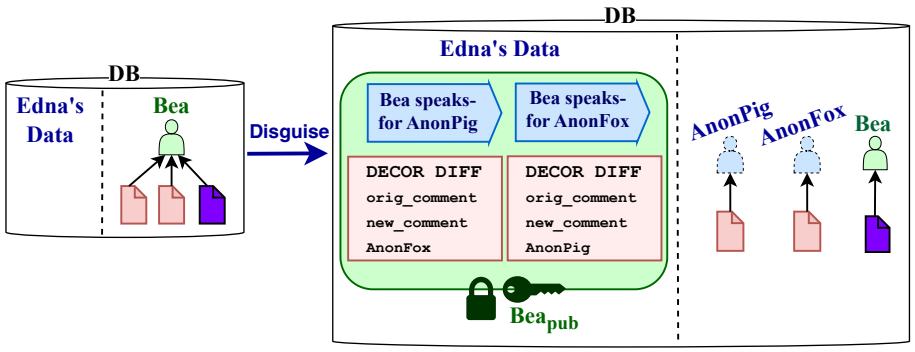


Figure 4-1: When Edna applies topic-based anonymization to Bea's comments on stories tagged "Star Wars" (red), these comments are decorrelated to pseudoprincipals ("AnonPig", "AnonFox"). Edna stores encrypted speaks-for records mapping Bea to their pseudoprincipals, and diff records containing the comments with modified foreign keys.

a pair of (original principal, pseudoprincipal) IDs and the pseudoprincipal's private key. Edna registers the pseudoprincipal with its public key to enable composition of disguises (§4.4). Edna then encrypts the diff and speaks-for records with the principal's key, and stores them in the database. Finally, Edna returns the disguise ID to the application. A client can use the disguise ID and the principal's credentials to reveal the transformation later.

To perform Bea's topic-based anonymization (Figure 4-1), Edna thus: (*i*) queries the database to fetch comments and votes by Bea affiliated with "Star Wars"; (*ii*) creates a pseudoprincipal (e.g., "AnonFox") for every "Star Wars"-tagged story that Bea commented on, and inserts it as a new user; (*iii*) modifies the database by rewriting comment foreign keys to point to the created pseudoprincipals, and removing Bea's votes on those stories; (*iv*) creates new speaks-for records that map Bea to the created pseudoprincipals and create new links in Bea's speaks-for chain; diff records containing Bea's votes on "Star Wars" stories; and diff records that document Bea's original ownership of "Star Wars"-tagged comments and the placeholder pseudoprincipal data; (*v*) encrypts the speaks-for and diff records with Bea's public key, stores them; and (*vi*) returns a unique disguise ID to the application.

Edna adds a *disguise table* and a *principal table* to the application database to store principals' disguised data. The disguise table contains lists of per-principal diff and speaks-for records encrypted with the principal's public key. The principal table is indexed by application user ID; each row contains the principal's public key, and a list of disguise table indexes encrypted with the public key. To store records for principal p, Edna (*i*) encrypts the records with p's public key; (*ii*) stores the ciphertext in the disguise table under index `idx`; (*iii*) encrypts `idx` (salted to prevent rainbow table attacks) with p's public key; and (*iv*) appends the encrypted `idx` to p's list of encrypted disguise tables indexes in the principal table.

This allows Edna to store records without needing access to the principal's private key, and to do so securely: the principal table adds a layer of indirection from user ID to encrypted records, so an attacker cannot link principals to their records. At reveal time, Edna can efficiently find

```
Reveal(disgID, uid, privkey):
 encrypted_disg_table_idxs := principal_table[uid]
 decrypted_disg_table_idxs :=
    decrypt(encrypted_disg_table_idxs, privkey)
 for idx in decrypted_disg_table_idxs:
   records = decrypt(disg_table[idx], privkey)
   for rec in records:
     if rec.disgID == disgID:
       // apply rec to application database
       // remove rec from disg_table
     else if rec.type == SPEAKS_FOR:
       // recursively reveal for pseudoprincipal
       // generated by another disguise
       Reveal(disgID, rec.pp_uid, rec.pp_privkey)
```

Figure 4-2: Pseudocode for revealing a disguising transformation while application principal `uid` exists. Recursive revealing (the `else` clause) walks the speaks-for chain to reveal composed records of pseudoprincipals created by other disguising transformations if necessary (§4.4).

disguised data for a given user by decrypting and using disguise table indexes in the principal table.

Disguising transformations may completely remove a principal from the application database. When this happens, Edna moves the corresponding list of encrypted disguise table indexes from the principal table to a *deleted principal table* indexed opaquely, e.g., by the public key. This removes the user ID from the database while allowing future reveal operations by the principal to find their disguise table indexes. Edna also stores a special type of remove diff record that contains the principal's ID and the principal's public key when a disguise removes a principal. This allows Edna to restore the principal to Edna's principal table if a user later reveals the disguise.

4.2 Revealing

To apply a reveal transformation, Edna first locates and decrypts the corresponding diff and speaks-for records using a disguise ID and the user's reveal credentials.

Edna's reveal procedure (Figure 4-2) first looks up all disguise records related to the provided reveal credentials via Edna's principal and disguise tables. Edna then applies diff records created for the `disgID` disguise transformation to the database, thus restoring the relevant application objects to their pre-disguised state.

To preserve referential integrity, Edna restores disguised data that was removed from tables in topologically-sorted order (again constructing the dependency graph). Edna then reveals any modifications, and finally performs recorrelations. In general, to reveal a diff record, Edna removes the placeholder row and inserts the original row (both of which are recorded in the diff record). However, for efficiency, Edna first checks if a placeholder row in diff records has the same identifiers (e.g., a primary key) as an original data row in the diff record, and if so, will update the relevant columns of the placeholder row instead.

Finally, Edna de-registers any pseudoprincipal who no longer has any associated disguised data, removing them from the principal table and the application's users table. Developers can configure Edna to also check for references to pseudoprincipals prior to removing them, and depending on the application's needs, configure Edna to delete, rewrite, or leave the references in place. After revealing, the disguised data is no longer needed, so Edna clears the corresponding diff and speaks-for records.

In the example, if Bea wants to reveal their "Star Wars" contributions, Lobsters invokes Edna with the disguise ID and Bea's password as reveal credentials. Edna uses the password to reconstruct Bea's private key, retrieve and decrypt Bea's records, and filter those records for those with the disguise ID. Edna then restores deleted votes and Bea's ownership of decorrelated comments.

4.2.1 Revealing in the Presence of Application Updates

Edna's revealing as described thus far may accidentally reveal data that ignores database changes applied since the time of disguise, such as application updates to undisguised data or schema migrations. To prevent this, Edna utilizes developer-provided information about important updates and schema migrations to apply them to disguised data prior to revealing it, ensuring that revealed data reflects the current state of the database. Prior to revealing the updated data, Edna will also perform consistency checks to guarantee adherence to internal database invariants, such as uniqueness constraints. Here, we describe how developers specify application updates and schema migrations; how Edna applies these updates to data to reveal; and Edna's consistency checks.

Applying Application Updates

In order for revealing to preserve application correctness, Edna must know about application updates that alter application invariants. Consider an example where a moderator periodically edits all posts to remove swear words, creating an implicit invariant that all posts created prior to the last moderation pass should contain no swear words. If a user's disguise removes their posts, then a moderation pass occurs, and then the user wants to restore their posts, Edna would not know about these moderation updates to posts of the application and incorrectly restore the original post content with swear words present upon reveal. However, Edna could correctly restore the post if it knew to remove the swear words prior to reveal.

Edna handles this situation by tracking applied application updates in a *replay log* of application update executables. As shown in Figure 4-3, the application invokes Edna when it performs operations that need to hold over disguised data when it is revealed—e.g., moderations—and Edna records these updates in its replay log (2). When revealing data, Edna applies, in order, every entry in the replay log added since the time of disguise to the disguise's diff records (3). Update executables map the original data and the placeholder data in diff records to correspond-

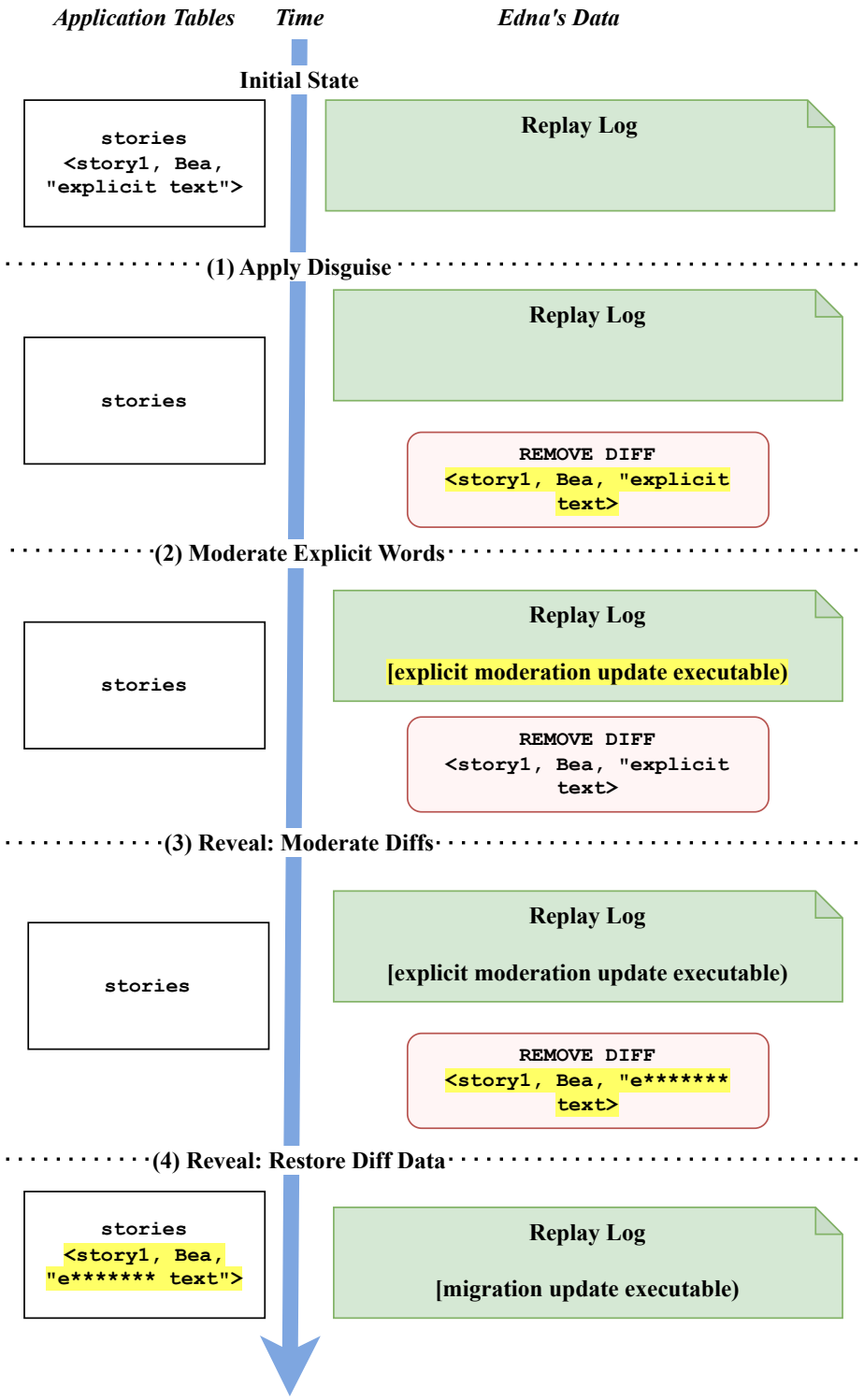


Figure 4-3: When the application applies updates like moderations of explicit words, it can invoke Edna to log the update. Edna then applies the update to disguised data in diff records prior to reveal. Yellow highlights changes to the application data or Edna's data.

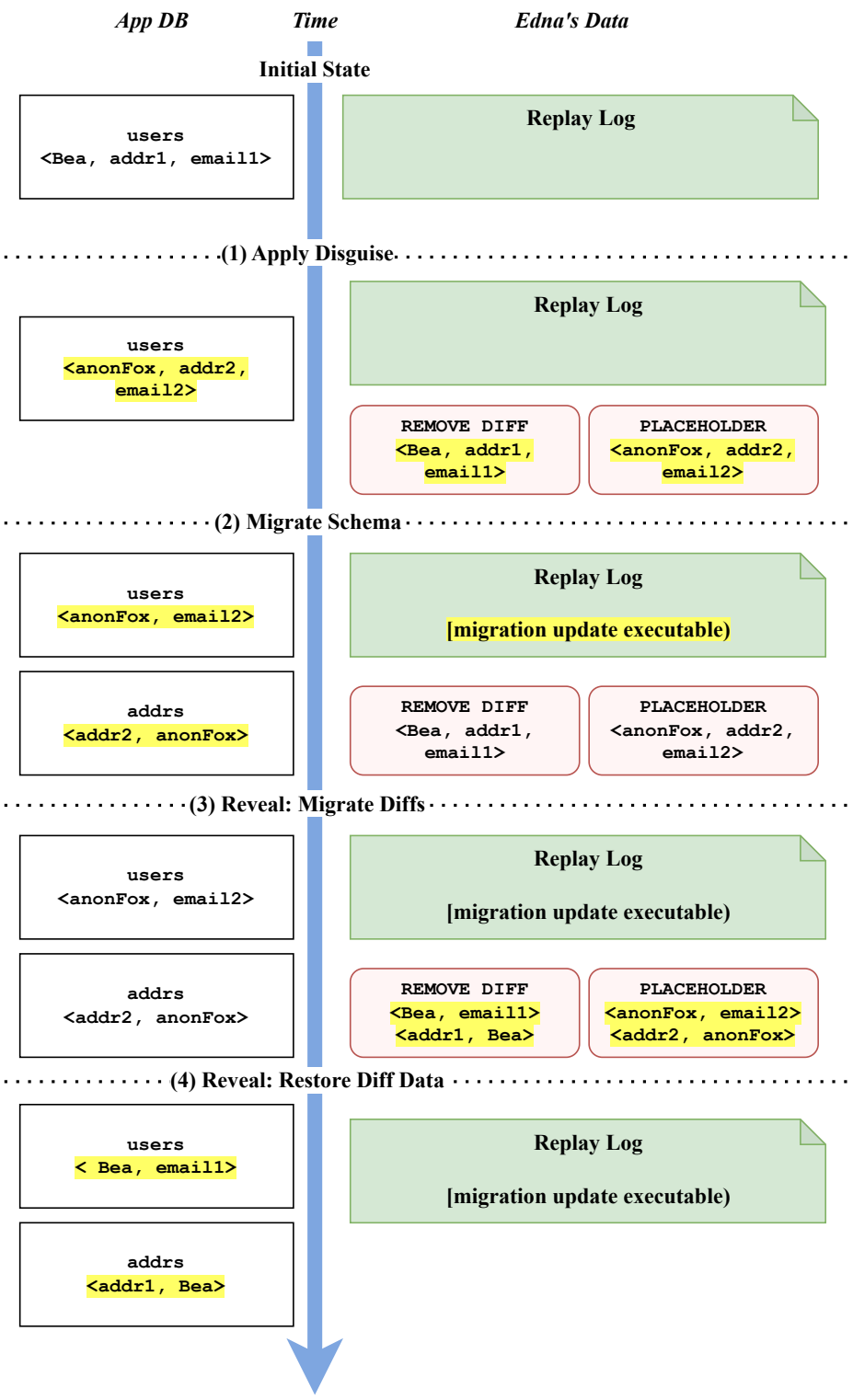


Figure 4-4: Similar to a moderations update, the application can notify Edna of a schema migration to apply to diff records prior to reveal. Yellow highlights changes to the application data or Edna's data. For simplicity, only the placeholder data of a pseudoprincipal row is shown; in reality, placeholder data exists as part of e.g., a decorrelation diff record.

ing updated-original and updated-placeholder data. After applying all updates to placeholder data, Edna reveals the updated rows in the diff record like normal (4).

In the swear words moderation example, Edna queries the replay log, sees the swear words moderation entry, and then applies the swear words moderation to the removed post data in the diff record. Only then does Edna restore the post with no swear words.[1]

Applying Schema Migrations

Applications also constantly undergo schema migrations to reorganize data or add new application features. When these occur, Edna must know how to manipulate any disguised data structured in the old database layout to match that of the current database.

For example, a developer of an application with a `users` table that contains rows with `username`s, `addr`s, and `email`s may choose to allow users multiple addresses. To do so, they would create a new `addrs` table, with a foreign key to the users table, and populate `addrs` using the address data in `users`. Finally, they would remove the `addr` column from `users`.

Figure 4-4 illustrates how applications performing this schema migration invoke Edna with an update executable encoding this schema migration to store in its replay log (2). As with application updates, schema migration executables map diff record data to migrated-original data and migrated-placeholder data (3), which Edna then respectively restores and removes (4). The migration update executable maps an original `users` data row in a diff record to both a row in `users` with username `uid` and a row in `addrs` that has a foreign key of `uid` to `users`, as shown in step (3). After applying the migration, Edna then restores these rows to the database (4). The executable also generates two rows for any placeholder `users` data row in a diff record and remove these from the database (4).

Consistency Checks for Internal Invariants

After applying any developer-specified migrations or updates since the time of disguising, Edna still needs to perform consistency checks on the data to reveal to ensure that revealing will not violate the database's integrity. These checks only allow revealing if the revealed data: (*i*) will still satisfy uniqueness and primary key constraints; (*ii*) will not overwrite modifications that occurred while data was disguised; and (*iii*) will maintain referential integrity.

For (*i*), Edna checks that *removed* disguised data is still removed from the database.

For (*ii*), Edna ensures that *modified* disguised data is in the same modified state and *decorrelated* disguised data is still affiliated with the same pseudoprincipal in the database using the new value stored in the diff record. By default, Edna performs checks at column-granularity. For example, an application database row can have two modified columns, but at the time of reveal, Edna finds that only one column still is at the modified state that Edna expects. The other

[1]Note that in this example, the diff record contains no placeholder data that replaced the removed post; in other scenarios such as decorrelation or modification, Edna would apply the logged update to placeholder data as well. The updated placeholder data would then be removed.

column must have been updated since the time of the disguise. Edna will then only reveal the one matching column that passed the check in order to avoid overwriting application updates, thus partially restoring the modified row. Developers can optionally specify that rows must be completely revealed, i.e., if any column fails the check, Edna should not reveal *any* column in the row.

To ensure (*iii*), Edna checks for the existence of all objects referenced by the data to reveal (e.g., a post referenced by a to-be-revealed comment).

Edna is conservative and will never reveal rows for which checks fail; the affected data remains disguised. For example, if a developer chooses not to register conflicting application updates or schema migrations, Edna's checks may fail, preventing disguised data from being revealed. Edna could flag encountered conflicts, giving the application a chance to fix them so a later reveal can pass the checks.

4.3 Shared Data

Many applications support shared data; in Lobsters, for example, messages between users are owned by both users. Edna's default semantics for shared data implement an ownership model inspired by a common treatment of messages as jointly owned. When a user disguises shared data, Edna decorrelates the data from the disguising user, but preserves the data and its association with other owners. Edna removes the data once all users have disguised it and all ownership links are to pseudoprincipals.

For instance, consider a Lobsters message between Bea and Chris as shown in Figure 4-5: after Bea disguises the message, the message is owned by Chris and a pseudoprincipal (1); if Chris then disguises the message, Edna removes it (2). Either owner can reveal the message, which restores the message to the database and recorrelates the revealing user (3); the other owner remains decorrelated as a pseudoprincipal until they also choose to reveal the message. Regardless of the reveal order, if all owners reveal the message, Edna returns the message to its original state (4).

Edna's Solution. Edna supports these joint ownership semantics with the following design, depicted in Figure 4-6. The first time any owner removes the data, Edna generates pseudoprincipals for each owner of the data, and creates a fully-decorrelated version of the shared data row (1). This decorrelated row is stored as a (plaintext) remove diff record in a *partially-removed* table created by Edna, indexed by the row's unique identifier columns (e.g., a primary key id).

When another owner's disguise removes the data (2), Edna stores a copy of the fully-decorrelated diff record for the disguise, diff records recording the associated pseudoprincipal row and foreign key rewrite for the owner, and a speaks-for record recording which pseudoprincipal the owner can speaks-for.

As shown in (2), if the owner also happens to be the last remaining owner (all other owners

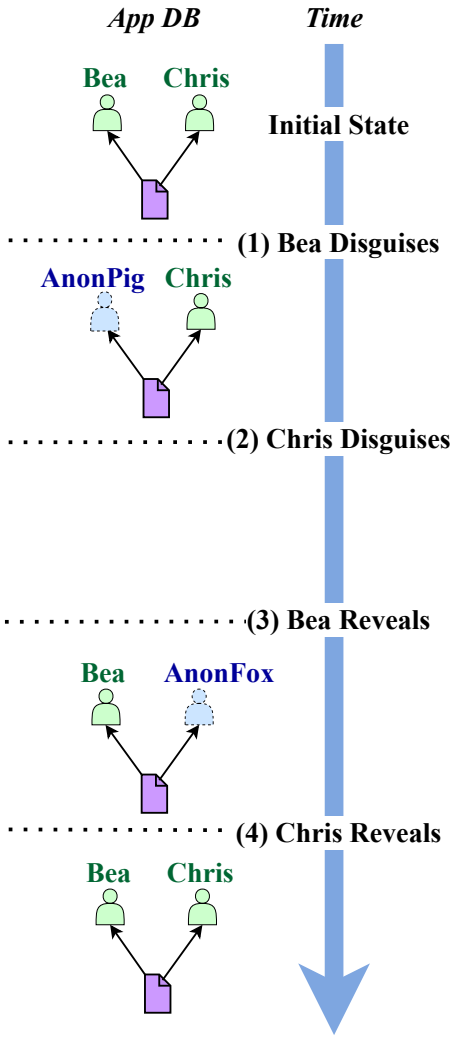


Figure 4-5: Edna supports joint ownership semantics, where shared data is not removed until all users have disguised their accounts. Owners can return in any order, and the message remains decorrelated from unrevealed owners.

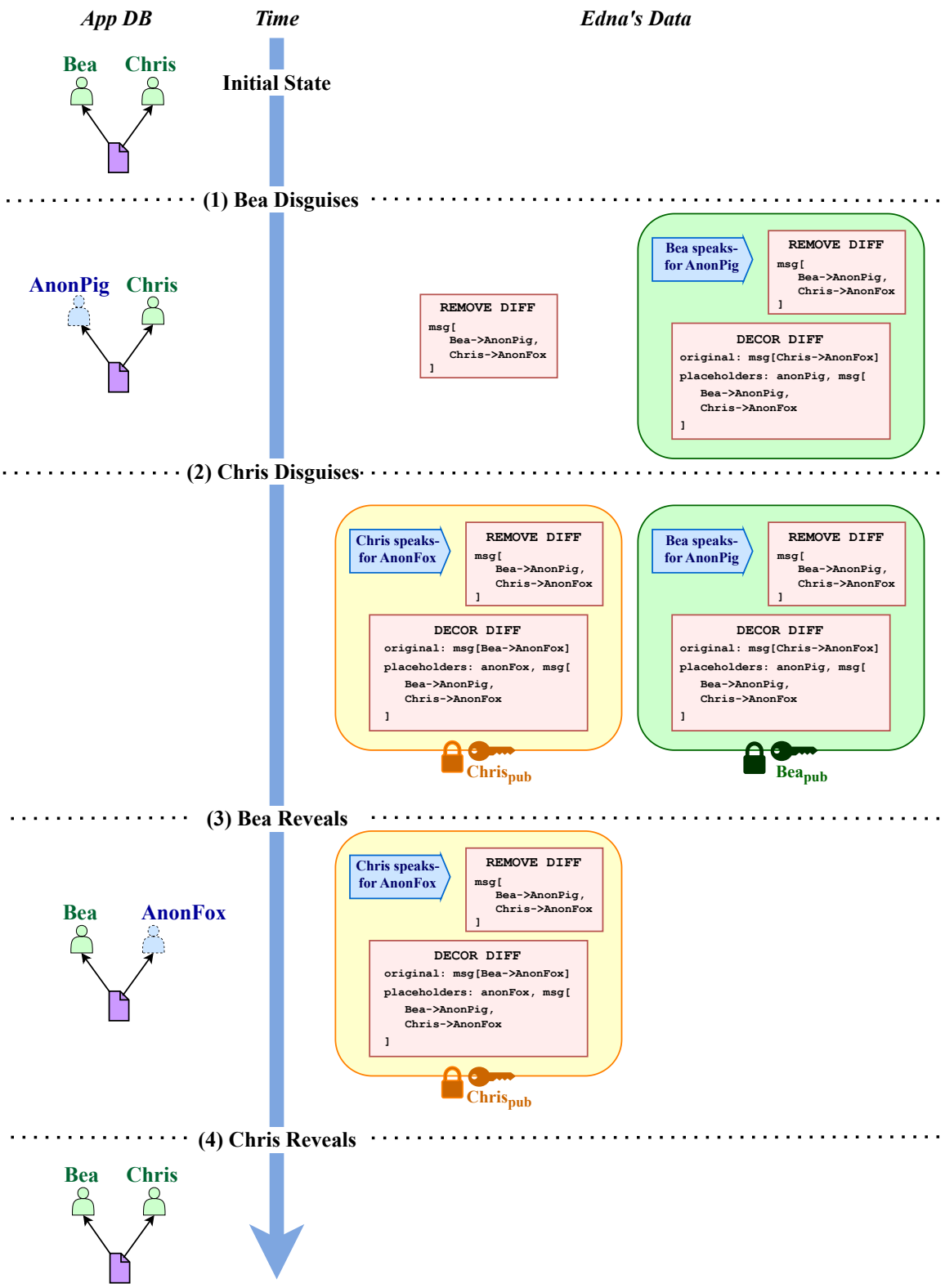


Figure 4-6: Edna implements joint ownership semantics by storing a fully decorrelated REMOVE diff for the shared data (modifications to the shared message data are depicted using [owner→pseudoprincipal] notation). Owners can only see pseudoprincipals for the other owners in their disguised data. This allows Edna to disguise and reveal in any order.

are pseudoprincipals, i.e., all other owners have removed the data), Edna removes the shared data row from the application database, and the fully-decorrelated diff record from Edna's partially-removed table.

Now, when *any* owner chooses to return (3), Edna first restores the fully-decorrelated row from the copy of the fully-decorrelated diff record in the owners's disguised data. Edna then rewrites the foreign key for the pseudoprincipal who currently owns the shared data to recorrelate with the owner, and removes that pseudoprincipal, revealing the other diff records like normal. If a subsequent owner reveals the shared data (4), Edna's consistency checks will prevent the reveal of the fully-decorrelated row (inserting the row will cause a duplicate row in the table). However, Edna will still rewrite the foreign key for that owner's pseudoprincipal to the original owner, and remove the pseudoprincipal from the database.

Thus, Edna can disguise and reveal shared data no matter the order in which its owners decide to remove or reveal it. If an owner never removes the shared data, they will continue to be correlated and have access to the data even if other owners remove it (and become pseudoprincipals); similarly, if an owner never restores the shared data, the data will remain forever owned by the owner's pseudoprincipal if other owners choose to restore it.

4.4 Composing Disguising Transformations

Edna supports composition of disguising transformations, which occurs when a transformation applies to data that Edna had previously disguised in some other way. Reasoning about composition of transformations can be broken down to reasoning about the composition of primitive operation pairs, e.g., remove after modify, or remove after decorrelation. Many pairs result in trivial composition: no operation can be composed after a remove (the data is gone), and any operation after a modify updates the data as expected.

However, operations after decorrelation results in more complex composition scenarios. For instance, decorrelation after decorrelation could occur if a user decorrelates some posts, after which an administrator decorrelates *all* posts. In this scenario, the administrator's disguising operation applies to pseudoprincipal-owned posts in the same way as it does to unmodified posts. This creates pseudoprincipals that can speak-for other pseudoprincipals. Edna uses the pseudoprincipal's registered public key to encrypt pseudoprincipal diff and speaks-for records, so Edna does not need to know its link to an original principal in order to encrypt and disguise its data.

Removal or modification after decorrelation also require special handling. For instance, a Lobsters user might first decorrelate some of their comments and then request to delete all their comments (e.g., by deleting their account). But the decorrelated comments are no longer linked to the original user; how can the deletion transformation find them? Edna addresses this question by accepting optional reveal credentials as part of the disguise operation. These credentials let Edna decrypt the user's previous diff and speaks-for records, find all pseudoprincipals corre-

sponding to that user, and apply disguising transformations on behalf of those pseudoprincipals as well as the user. Edna uses pseudoprincipal private keys in decrypted speaks-for records as credentials to recursively find pseudoprincipals from multiple decorrelations.

Out-of-Order Reveals. Edna must also handle reveals of transformations in any order. As before, many scenarios are straightforward: revealing removals is trivial (data can only be removed and restored once), and revealing modified data simply restores the original (subject to consistency checks). Handling out-of-order reveals of multiple decorrelations presents the greatest challenge. Edna's semantics enforce that data that is decorrelated multiple times will not be recorrelated until all disguises are removed. For example, as shown in Figure 4-7 (left-hand side), if Bea separately decorrelates their comments on "bears" and "Star Wars" posts, then later reveals the "bears" posts, they might want Ewok-related comments (tagged both "Star Wars" and "bears") to remain disguised, even though they were initially disguised under the "bears" transformation. To support this, Edna maintains a chain of speaks-for records that represent speaks-for relationships between pseudoprincipals. All reveal operations walk the full speaks-for chain to reveal all necessary records (cf. Figure 4-2).

Furthermore, if reveal operations perform recorrelations out of order, Edna removes an intermediate link in the speaks-for chain. To describe how this works, take the example of two disguises that decorrelate comments on "bears" and "Star Wars" posts respectively (Figure 4-7).

(1) First, the "bears" decorrelation rewrites Ewok posts to pseudoprincipal P_1.

(2) Next, the "Star Wars" decorrelation composed on top decorrelates Ewok posts from P_1 to pseudoprincipal P_2. At this point, Edna has speaks-for records for Bea that encode a speaks-for chain from Bea$\rightarrow P_1 \rightarrow P_2$.

(3) If Bea first reveals the "bears" anonymization—the first applied disguise—on their data, Edna finds all accessible speaks-for records given Bea's reveal credentials (cf. Figure 4-2), and constructs Bea's speaks-for chain as a graph of principal-to-principal edges (namely Bea$\rightarrow P_1 \rightarrow P_2$). Edna finds that P_1's "Ewok" posts no longer exist in the database (they belong to P_2 due to the composed "Star Wars" disguise), and thus does not restore "Ewok" posts to Bea. Edna does still remove pseudoprincipal P_1 in the process of restoring "bear" diff records, and once done, clears all "bear" 'diff records from its encrypted disguise table. Importantly, however, Edna still retains the speaks-for record for Bea$\rightarrow P_1$, since P_1 still has associated disguised data (namely a speaks-for record to P_2)!

(4) Now, if Bea reveals the second applied disguise—"Star Wars"—Edna finds diff records for decorrelated "Ewok" posts whose original rows have P_1 as owner. However, Edna cannot reveal the diff record directly and restore the decorrelated posts to P_1, as this will fail referential integrity consistency checks, and also result in incorrect composition semantics. Instead, Edna walks the speaks-for chain (Bea$\rightarrow P_1 \rightarrow P_2$) backwards from P_2 to determine the *next valid* user in the chain who speaks-for P_2, which in this case is Bea. Validity is determined by whether the user is a natural principal or a pseudoprincipal yet to be recorrelated. If a pseudoprincipal has

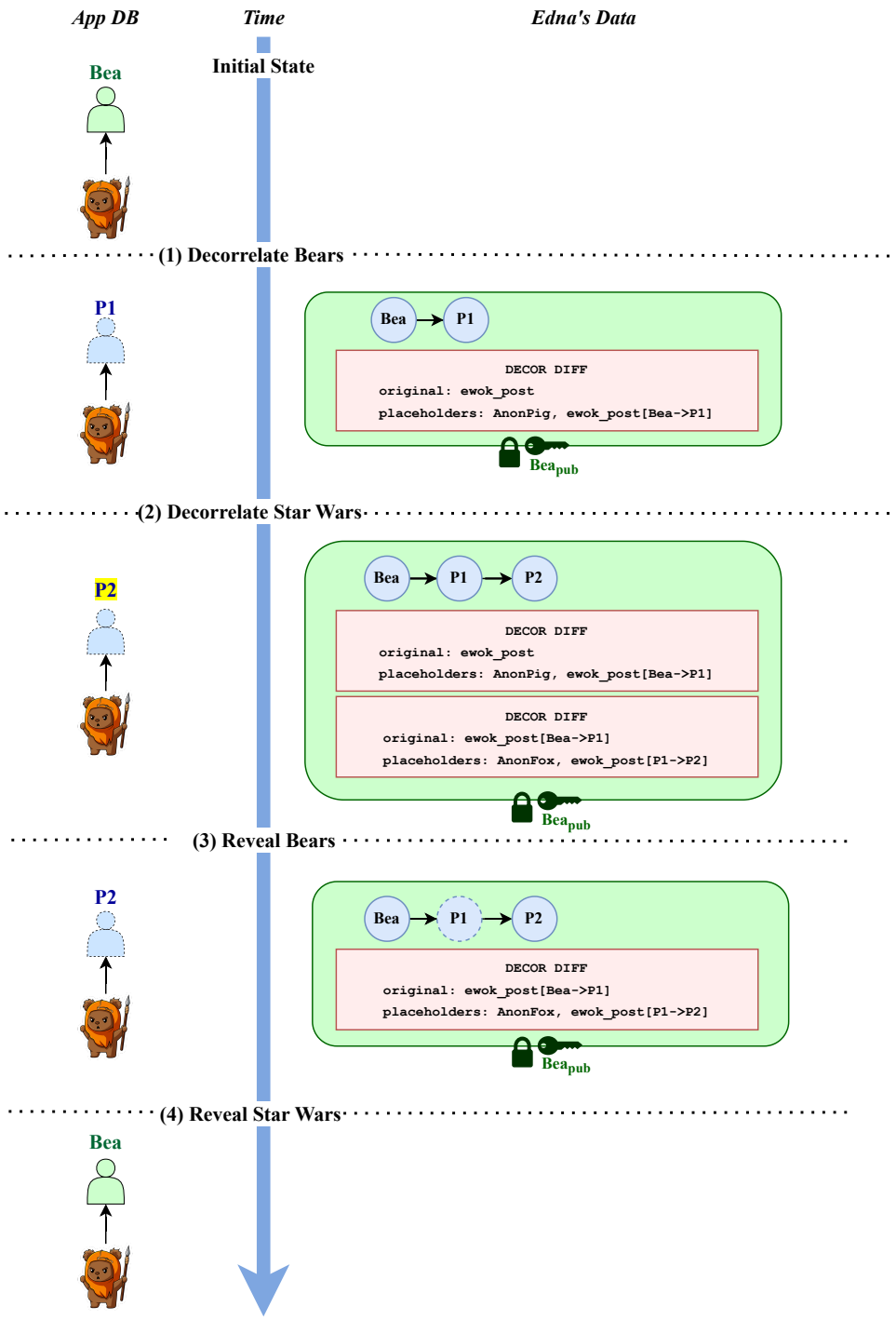


Figure 4-7: Edna allows multiple decorrelations to compose, and can reveal them in any order such that the data remains decorrelated until the user reveals *all* applied decorrelations. The speaks-for chain encoded in speaks-for records (shown in blue) allows Edna to handle recorrelation of intermediate pseudoprincipals. When Bea reqests reveal of their "Star Wars" posts (4) after revealing "bears" posts (3), Edna walks the speaks-for chain backwards from P_2 to find the latest principal that can speak-for P_2 and has not yet been recorrelated. Thus, Edna restores ownership back to Bea.

a decorrelate diff record from any disguise still present in Bea's disguised data, Edna knows the pseudoprincipal has not yet been recorrelated.

After determining Bea is the next valid user in the speaks-for chain to P_2, Edna rewrites the "Ewok" diff record to reference Bea instead of P_1, and then restores the diff record like normal. Finally, Edna deletes the diff records for the "Star Wars" disguise, the speaks-for records for P_2 (which no longer has any associated disguised data), and the speaks-for records for P_1 (which also has no associated disguised data after P_2 is removed). This implicitly truncates the speaks-for chain to simply be Bea. In general, Edna enforces the invariant that the chain only truncates at link L once all pseudoprincipals recursively generated in the chain beyond L have been recorrelated.

4.5 Authenticating As Pseudoprincipals

As described so far, if Bea wanted to modify a decorrelated "Star Wars" comment, they would have to reveal the comment, edit it using their normal credentials, and then redisguise the comment. Edna applications can also let users modify decorrelated records without the reveal step. To support this, an application accepts reveal credentials along with a modification request. Edna uses these credentials to validate that the user speaks-for a specific pseudoprincipal by walking the speaks-for chain (cf. Figure 4-2) and ensuring that the user can access a speaks-for record to the pseudoprincipal. After validating the user's request, Edna updates the database with the modification. check

4.6 Design Limitations

XXX moved api here?

Disguises. Edna assumes that all desired disguises are captured with the three primitive operations (remove, modify, and decorrelate), and that Edna's generation specifications cover all desired values for modifications or pseudoprincipals. However, this means that placeholder data values cannot take as input external state (e.g., the current state of the database), to, for example, add noise dependent on the number of rows in the database. We can imagine expanding Edna's generation specifications to take as input a database query to generate new values.

Furthermore, disguising transformations may affect results of data processing on the database (e.g., aggregates over the number of users). Edna currently expects the developer to correctly handle these scenarios, and ensure that any aggregate results or placeholder data does not violate application correctness.

API. Edna's API assumes that:

1. the application uses a relational database;

2. rows to disguise have direct foreign key relationships to a users table, where each user corresponds to a row of that table;
3. all rows to disguise are owned by one or more principals; and
4. all rows can be uniquely identified (e.g., via primary key).

Applications that do not satisfy these assumptions—e.g., because they have complex ownership chains or use a NoSQL database—could be supported with extensions to Edna's design. Like in DELF [17], Edna could support multiple data models by requiring developers to provide (*i*) a model of data as objects with dependency edges to other objects, and (*ii*) instantiations for update/delete/insert operations on these objects. To handle indirect data ownership, Edna can use an approach similar to K9DB's data ownership graph [19]. And if no user owns a data item, Edna could perhaps refuse to disguise it, and flag the developer to review the disguise specification. hmm better idea here? To address the unique identification requirement, Edna could add unique IDs for every data object, so Edna can refind the object when revealing. However, this method requires more invasive changes to application data.

Edna's Replay Log. Edna's replay log also faces some limitations. First, developers must do additional work to add hooks to invoke Edna when any major application change or schema migration occurs. Invoking these hooks and applying these updates during reveal also adds additional computation work; we measure these costs in §7.2.1.

Furthermore, updates in replay logs only apply to diff records, which contain the actual data changes, and not speaks-for records; Edna assumes that the identifiers for principals in speaks-for records that encode the speaks-for chain remain consistent as the database changes, allowing Edna to use its composition techniques with speaks-for chains (§4.4) to reveal multiple disguises in any order.

should this be here or part of spec in §3? Finally, Edna assumes that either (*i*) update executables are pure functions deterministic in their input (placeholder or disguised rows), and do not have side effects; or (*ii*) any nondeterministim or side effects of an update executable will maintain application correctness. For a counterexample, take an update that calculates the current vote count of a post to store as a post's `vote_count` attribute, and say that a disguise has removed both a user's post and its votes. The update thus depends on other disguised rows, namely `vote` rows. When Edna reveals the disguise, Edna will first reveal the diff records for posts before revealing those for votes in order to maintain referential integrity. But when applying the `vote_count` update to post diff records, Edna will find 0 votes for the post, because the votes have not yet been restored. Edna only supports this update if this is correct application behavior (scenario (*ii*)).

One potential solution would be for developers to indicate the data dependencies of each update, so Edna could know to restore votes before posts; however, this requires Edna to disable foreign key checks and carefully check for referential integrity after restoring posts, since restoring votes first can now inadvertently leave dangling pointers. Or perhaps developers could

schedule cron jobs to habitually perform the update to "fix up" any rows that have been revealed since the update; however, this only works for an idempotent update, as it should not modify already-modified rows again.

4.7 Security Discussion

Edna's design achieves confidentiality of disguised data between the time of disguising and revealing, its key goal. Some aspects of Edna's design help make Edna practical and deployable without major application modifications, but give up stronger security in exchange for usability.

Under Edna's threat model, Edna achieves:

1. confidentiality of disguised data, via encrypting disguised data using asymmetric encryption, so only the owning user's private key can reveal it;
2. confidentiality of which encrypted disguised data belongs to which user, via opaque, encrypted indexing to reference a user's disguised data; and
3. reduced linkability between parts of a user's data, via splitting data ownership among pseudoprincipals.

However, an attacker sees all application database content and code, and Edna's disguise, principal, and deleted principal tables. Thus, what the attacker learns includes:

1. any undisguised data in the application database;
2. the active principals that have disguised data, via Edna's principal table;
3. the pseudoprincipals currently registered, from Edna's principal table and the application DB;
4. the number of deleted principals, via the size of the deleted principal table;
5. the amount of disguised data in Edna; and
6. the disguise specifications, from application code.

Edna provides decorrelation with pseudoprincipals to ease integration with existing applications, even though pseudoprincipals (and their mere existence) can reveal information to the attacker. Pseudoprincipals preserve application data and referential integrity, ensuring that e.g., every post always has an author, or that vote counts on posts remain unchanged, without requiring the developer to handle special cases of deleted users and orphaned data. However, this necessarily leaves information in the database.

Similarly, leveraging the application database to store disguised data increases Edna's practicality as it reuses existing server-side storage and avoids burdening users with managing their disguised data, but leaves potentially exploitable metadata available to attackers. An attacker could leverage pseudoprincipal groupings (e.g., a pseudoprincipal owning posts in both "CMU 2018" and "BayArea" topics), undisguised data (e.g., comments signed with the user's name), and Edna metadata (e.g., that some anonymous user has more disguised data than another, as Edna stores disguised data without padding for efficiency) to infer the identity of the original owning principal.

Finally, Edna makes no guarantees for users who actively use disguised data after compromise (e.g., by revealing or editing decorrelated data): after an attacker compromises the application at time t, they can harvest private keys that clients provide after t. However, Edna always protects users' disguised data if they remain inactive.

The attacker never has access to a user's private key unless the user actively provides their credentials. The attacker also cannot access the private key of any pseudoprincipal because it is in an encrypted speaks-for record. If an application uses password-based reveal credentials, Edna guarantees security equivalent to the security of the user's password.

Chapter 5

Implementation

Edna's prototype consists of [lyt: TODO]k lines of Rust. Edna also provides an API server that exposes a JSON-HTTP API that applications can use to invoke Edna. Our Edna prototype supports MySQL. check.

5.1 Secure Record Storage

When encrypting diff and speaks-for records, Edna appends a random nonce to the record plaintext to prevent known-plaintext attacks. It then generates a new public/private keypair for x25519 elliptic curve key exchange. Using the newly created private key and the principal's public key, Edna performs the x25519 elliptic curve Diffie-Hellman ephemeral key exchange to generate a shared secret. Edna encrypts the record data with the shared secret, and saves the ciphertext along with the freshly generated public key (required to decrypt the data given the principal's private key). This public key algorithm lacks key anonymity, so an attacker can determine which records belong to the same principal, but this is not fundamental [7].

5.2 Reveal Credentials

Our prototype supports two forms of reveal credentials: (*i*) private keys; or (*ii*) principals' passwords, and recovery tokens in case they forget their passwords. If an application chooses to use the latter, it provides the principal's password to Edna upon user registration. Our prototype uses a variant of Shamir's Secret Sharing [62] to generate three shares from the private key, any two of which can reconstruct the private key. Shares are $(x, f(x) \bmod p)$ tuples, where $f(x) = \texttt{privkey} + \texttt{rand} \cdot x$ and $p > \texttt{privkey}$ is a known prime. One share derives x from the user's password using a Password-Based Key Derivation Function (PBKDF) [36]. Edna stores the resulting $f(x)$ half of the share, allowing Edna to derive one full share from the password. Edna returns the second full share as a recovery token and stores the third full share. Edna can combine this third share with the recovery token or a full share derived from the password to

recover the private key.

The PBKDF ensures that Edna cannot guess the password-derived value with dictionary and rainbow table attacks [73], and that Edna cannot brute force the recovery token.

Password-based secret-sharing is only one possible implementation for backup secrets; Edna could also support password-based backup secrets by e.g., storing an version of the private key encrypted with the user's password.

5.3 Application Updates Replay Log

Edna represents application update executables as Rust functions from database objects (<table, row> pairs) to a new set of database objects. Edna stores the log as a vector of updates function pointers with timestamps, and persists the pointers to disk. This approach requires recompiling Edna to add new update functions; an alternative approach could put the functions into separate executables (and Edna would store executable names instead of function pointers). In this alternative, Edna would invoke these executables upon reveal with serialized input rows, and deserialize the output.

5.4 Batching

For efficiency, Edna batches the deletion of rows from the same table, and the creation of pseudoprincipals upon disguise. Edna also batches the deletion of pseudoprincipals during reveal. However, Edna does not batch insertion of other table rows during reveal, as each row's foreign key dependencies and relevant application updates need to be applied to each row; however, with careful implementation, Edna could support batching of all insertions. TODO should we batch other tables? it's much more complex. Edna performs decorrelation and modification operations individually for each row during both disguise and reveal, as different rows are updated to different values.

5.5 Concurrency

Edna runs disguising and revealing transformations in transactions, providing serializable isolation to application users. If a query within a transformation fails, the entire transformation aborts (returning an error to the application). Edna provides an option to run long-running transformations that touch large amounts of data (e.g., anonymization of all users' posts) without a transaction, at the expense of clients potentially observing intermediate states.

Chapter 6

Case Studies

This section evaluates Edna's ability to add new data-redacting features to several applications; §7 evaluates the effort needed to do so and the resulting performance.

We add disguising and revealing transformations based on the motivating examples in §1 to three applications—Lobsters [1], WebSubmit [60], and HotCRP [37].

6.1 Lobsters

Lobsters is a Ruby-on-Rails application backed by a MySQL database. Beyond the previously-mentioned stories, tags, etc., Lobsters also contains moderations that mark inappropriate content as removed. We added three disguising transformations: account deletion with return; account decay, i.e., automatic dissociation and protection of old data; and topic-specific throw-away accounts.

GDPR-compliant account deletion (*i*) removes the user account; (*ii*) removes information that's only relevant to the individual user, such as their saved stories; (*iii*) modifies story and comment content to "[deleted content]"; (*iv*) decorrelates private messages; and (*v*) decorrelates votes, stories, comments, and moderations on the user's data. This preserves application semantics for other users—e.g., vote counts remain consistent even after account deletion, and other users' comments remain visible—while protecting the privacy of removed users. Important information such as moderations on user content remains in the database, and Edna recorrelates it if the user restores their account. After Edna applies the disguising transformation, Lobsters emails the user a URL that embeds the disguise ID. The user can visit this URL and provide their credentials to restore their account.

The **account decay** transformation protects user data after a period of user inactivity. We added a cron job that applies account decay to user accounts that have been inactive for over a year. This (*i*) removes the user's account; (*ii*) removes information only relevant to the user, such as saved stories; (*iii*) and decorrelates votes, stories, comments, and moderations on the user's data by associating them with pseudoprincipals. Lobsters sends the user an email which

informs them that their data has decayed, and includes a URL with an embedded disguise ID that can reactivate or completely remove the account if credentials are provided.

Finally, topic-based throwaway accounts via **topic-based anonymization** enable users to decorrelate their content relating to a particular topic. As per §3.1, this disguises contributions associated with the specified tag by (*i*) decorrelating tagged stories and comments associated with tagged stories, and (*ii*) removing votes for tagged stories. Again, Lobsters sends the user an email with links that allow reclaiming or editing these contributions.

With Edna and its support for composing disguising transformations, users can delete accounts that have been decayed or dissociated into throwaways, and can later reveal them.

Application Updates and Schema Migrations check We also added support for three recent application updates and schema migrations applied by the Lobsters developers in 2023-2024.

The first is an application update and schema migration which performs URL normalization [8] on the `url` column of all rows in the `stories` table, and then stores the normalized URL text in a new `normalized_url` column.

The second changes the schema by adding a `show_email` attribute to the `users` table automatically set to `false`.

Finally, the third creates a new `story_texts` table which stores the `title`, `description`, and `story_cache` attributes of rows from `stories`. This enables search over story content. The `story_cache` column is then removed from `stories`.

6.2 WebSubmit

We integrated Edna as a Rust library with WebSubmit [60]. WebSubmit is a homework submission application used at Brown University, and its schema consists of tables for lectures, questions, answers, and user accounts. Clients create an account, submit homework answers, and view their submissions; course staff can also view submissions, and add/edit questions and lectures. The original WebSubmit retains all user data forever. We added support for two disguising transformations: GDPR-compliant user account removal with return, and instructor-initiated answer anonymization, which protects data of prior years' students by decorrelating student answers for a given course. These transformations allow instructors to retain FERPA-compliant [68] answers after the class has finished. With Edna, students can delete their accounts or access and view their answers even after class anonymization, and can always restore their deleted accounts, including restoring them to anonymized state.

6.3 HotCRP

HotCRP is a conference management application whose users can be reviewers and/or authors. HotCRP's schema contains papers, reviews, comments, tags, and per-user data such as watched

papers and review ratings [37]. HotCRP currently retains past conference data forever and requires manual requests for account removal [38]. We wrote two disguise specifications for HotCRP: conference anonymization to protect old conference reviews, and GDPR account removal with return.

Conference anonymization is invoked by PC chairs after the conference and decorrelates users from their submissions, reviews, comments, and per-user data such as watched papers. User accounts remain in the database with no associated data. Conference anonymization protects users' data after the conference; with Edna, users can come back to view or edit their anonymized reviews and comments.

Account removal (*i*) removes the user's account; (*ii*) removes information only relevant to the user, such as their review preferences; (*iii*) removes their author relationships to papers; and (*iv*) decorrelates the remainder of their data, such as reviews. Decorrelating a review removes its association with the reviewing user, but importantly keeps the review itself around to preserve utility for others (e.g., the PC and the authors of the reviewed paper). With Edna, users can remove their accounts even after conference anonymization has taken place, and can always restore their accounts.

Chapter 7

Evaluation

Update with numbers from Google Cloud Fix figures

Our evaluation seeks to answer six questions:

1. How much developer effort and application modification does Edna require? (§7.1)
2. How expensive are common application operations, as well as disguising, revealing, and operations over disguised data with Edna? (§7.2)
3. What overheads does Edna impose, and where do they come from? (§7.3)
4. How does the effort required to implement Edna's functionality in a related system (Qapla [45]), and its performance, compare with using Edna? (§7.4)
5. What is the performance impact of composing Edna's guarantees with those of encrypted databases? (§7.5)
6. Which categories of application updates and schema migrations can Edna support? (§7.6)

We compare Edna to three alternative settings: (*i*) a manual version of each disguising transformation that directly modifies the database (e.g., via SQL queries that remove data), which lacks support for revealing and does not support composition of multiple transformations; (*ii*) an implementation of disguising and revealing in Qapla [45] using Qapla's query rewriting and access control policies; and (*iii*) an integration of Edna with CryptDB [53], an encrypted database.

All benchmarks run on a Google Cloud `n1-standard-16` instance with 16 CPUs and 60 GB RAM, running Ubuntu 20.04.5 LTS. Benchmarks run in a closed-loop setting, so throughput and latency are inverses. We use MariaDB 10.5 with the InnoDB storage engine atop a local SSD.

7.1 Edna Developer Effort

We evaluate the developer effort required to use Edna by measuring the difficulty of implementing the disguising and revealing transformations in our three case studies. This took one person-day per case study for a developer familiar with Edna but unfamiliar with the applications.

A developer supporting these transformations must first add application infrastructure to allow users to invoke them and notify users when they happen. This is required even if the developer were to implement transformations manually without Edna. These changes add 179 LoC of Ruby to Lobsters (160k LoC), and 312 LoC of Rust to the original WebSubmit (908 LoC). They implement HTTP endpoints, authorization of anonymous users, and email notifications.

A developer using Edna also writes disguise specifications and invokes Edna. Lobsters' disguise specifications are written in 518 LoC, WebSubmit's in 75 LoC, and HotCRP's in 357 LoC (all in JSON). The specification size is proportional to schema size and what data each application disguises.

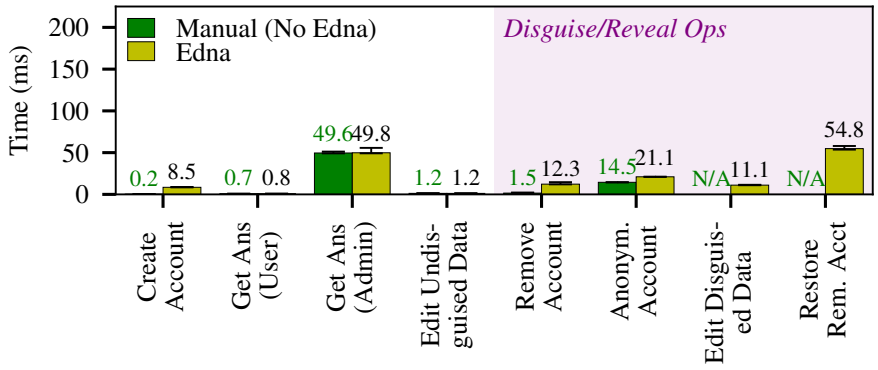
check Developers also notify Edna of application updates and schema migrations that may affect disguised data that could be later revealed. Notifying Edna of three different application updates and schema migrations for Lobsters required adding one line of code per update to register the update executable with Edna, after the bulk update is performed. Developers already write these bulk application updates (e.g., Lobsters implements these updates as Ruby Active Record Migrations [57]). Our Rust/SQL implementation of the three bulk updates required xxx LoC, and the corresponding update executables for disguised data required xxx LoC. The bulk update for URL normalization reuses the update executable which performs URL-normalization as a subroutine. The bulk updates for the other two schema migrations required writing separate update executables that operate on individual rows, instead of on entire tables.

xxxx what conclusion? Thus, the developer effort required to use Edna—writing Edna specifications, and invoking Edna—is small, even though these applications were not written with Edna in mind.

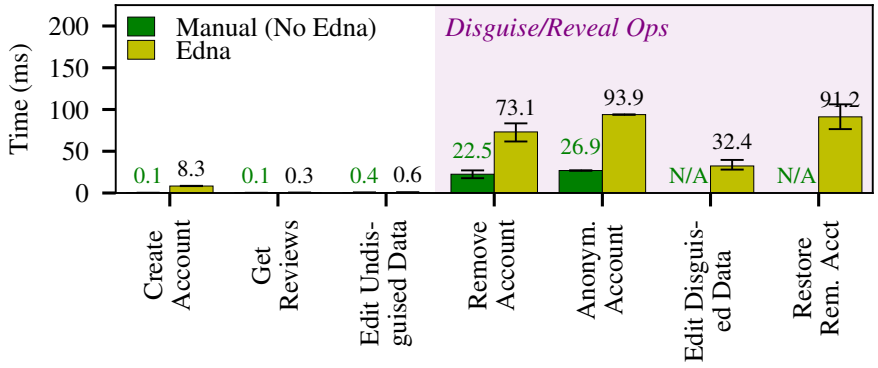
7.2 Performance of Edna Operations

We now evaluate Edna's performance using WebSubmit, HotCRP, and Lobsters (§**??**). We measure the latency of common operations, disguising transformations, and operations over disguised data enabled by Edna (e.g., account restoration and editing disguised data). The three applications do not create new data that references pseudoprincipals, but to fully capture any overheads we configure Edna to nevertheless run the checks for lingering pseudoprincipal references on revealing. A good result for Edna would show no overhead on common operations, competitive performance with manual disguising, and reasonable latencies for revealing operations only supported by Edna (e.g., a few seconds for account restoration)

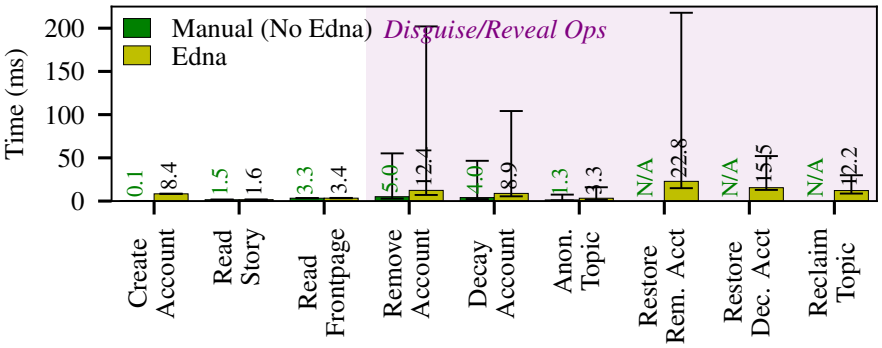
WebSubmit. We run WebSubmit with a database of 2k users, 20 lectures with four questions each, and an answer for each question for each user (160k total answers). We measure end-to-end latency to perform common application operations (which each issue multiple SQL queries), as well as disguising and revealing operations when possible (revealing operations are impossible in the baseline). Figure 7-1a shows that common operations have comparable latencies with and without Edna. Edna adds 9ms to account creation; and disguising and revealing



(a) WebSubmit (2k users, 80 answers/user).



(b) HotCRP (80 reviewers, 3k total users, 200–300 records/reviewer).



(c) Lobsters (16k users, Zipf-distributed data/user).

Figure 7-1: Edna adds no latency overhead to common application operations and modestly increases the latencies of disguising operations compared to a manual implementation that lacks support for revealing or composition. Bars show medians, error bars are 5^{th}/95^{th} percentile latencies.

operations take longer in Edna (13.1–53.2ms), but allow users to reveal their data and take less developer effort.

HotCRP. We measure server-side HotCRP operation latencies for PC members on a database seeded with 3,080 total users (80 PC members) and 550 papers with eight reviews, three comments, and four conflicts each (distributed evenly among the PC). HotCRP supports the same disguising transformations as WebSubmit, but PC users have more data (200–300 records each), and HotCRP's disguising transformations mix deletions and decorrelations across 12 tables.

Figure 7-1b shows higher latencies in general, even for the manual baseline, which reflects the more complex disguising transformations. Edna takes 63.8–84.6ms to disguise and reveal a PC member's data, again owing to the extra cryptographic operations necessary. HotCRP's account anonymization is admin-applied and runs for all PC members, so its total latency is proportional to the PC size. With 80 PC members, this transformation takes 6.8s, which is acceptable for a one-off operation. As before, Edna adds small latency to common application operations, and 9ms to account creation.

Lobsters. We run Lobsters benchmarks on a database seeded with 16k users, and 120k stories and 300k comments with votes, comparable to the late-2022 size of production Lobsters [1]. Content is distributed among users in a Zipf-like distribution according to statistics from the actual Lobsters deployment [32], and 20% of each user's contributions are associated with the topic to anonymize. The benchmark measures server-side latency of common operations and disguising/revealing transformations.

The results are in Figure 7-1c. The median latencies for entire-account removal or decay are small (9.7–13.4ms for Edna, and 4.0–5.2ms for the baseline), since the median Lobsters user has little data. Revealing disguised accounts takes 13.1–17.6ms in the median. Highly active users with lots of data raise the 95th percentile latency to 100–180ms for disguising and 45–80ms for revealing. Topic anonymization touches less data and is faster than whole-account transformations, taking 3.6ms and 13.1ms for the median user to disguise and reveal, respectively.

Summary. Edna necessarily adds some latency compared to manual, irreversible data removal, since it encrypts and stores disguised data. However, most disguising transformations are fast enough to run interactively as part of a web request. Some global disguising transformations—e.g., HotCRP's conference anonymization over many users—take several seconds, but an application can apply these incrementally in the background, as in Lobsters account decay.

7.2.1 Performance of Reveals with Database Changes

We next look at how operating in the presence of schema migrations or conflicting application updates changes with Edna, and how it affects Edna's reveal performance.

First, we look at the cost of invoking Edna to record updates when the application performs updates or schema migrations in Lobsters. Applying these changes themselves equates to modifying all 120k stories (e.g., URL normalization) and/or changing the database schema. The

URL normalization application update takes 5.63s, the longest of all updates. This matches the optimized bulk update time of 6s in the deployed Lobsters application, which batch updates all stories with the normalized URL. The unoptimized version that updates stories one-by-one takes 1789s.

Adding the `show_email` attribute to the `users` table takes 82ms, and creating and populating the `story_texts` table, and removing a column from the `stories` table takes 4.93s.

Invoking Edna's hook to insert a new application update in Edna's replay log takes on average 0.58ms, a negligible amount compared to the time to perform the update/migration. This includes the time to persist the update (one database insert query) and add it to Edna's in-memory update replay log.

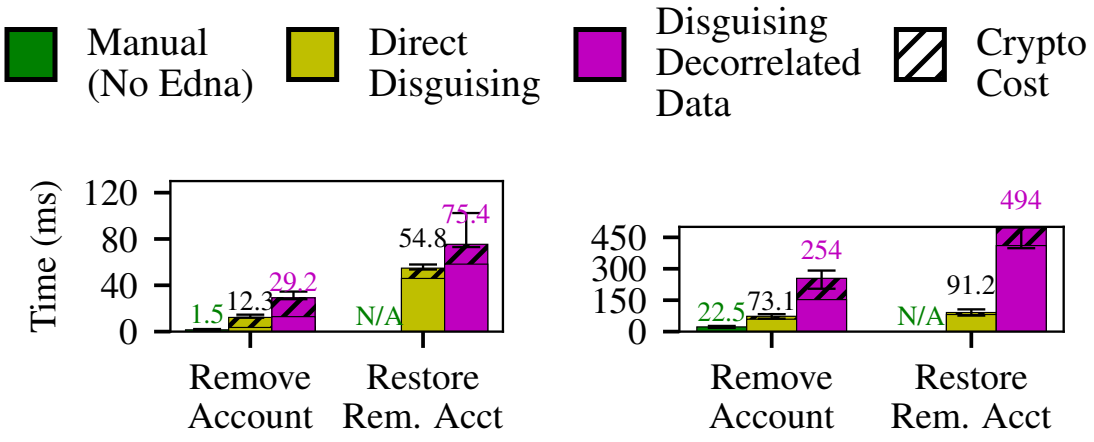
We next evaluate the latency of reveal operations in Lobsters when the global database changes described in §6.1 have been applied after reveal. The effects of performing updates increases with the amount of disguised data a user wants to reveal. For our support updates, the main overheads come from revealing disguised stories, which increases by 0.5–0.8ms per disguised story. Our chosen updates either affect diff records containing `users` or `stories` data. URL normalization and reorganizing into `story_texts` both affect `stories`, and take on average 0.4–0.5ms and 0.02ms respectively when applied to the original and placeholder rows for a single `stories` diff record. The main cost of URL normalization comes from initializing a URL normalizer object (≈0.3ms). we could amortize this by putting the normalizer in Edna or something, but this would be cheating... or batching all story diff records into one update. Restoring a stories diff record also requires an additional query to reveal a `story_texts` table row, incurring a 0.1–0.3ms cost per story. (Revealing other table diff records adds only the cost of checking whether to apply the updates, which takes 0.01–0.02ms per diff record). In total, we observe that users with very few (e.g., < 5 posted stories) incur negligible additional overheads , but users with lots of data incur moderate overheads proportional to the number of their stories (e.g., ≈400ms to reveal an account with 600-700 stories).

there's extra overheads for the updates test simply because db queries take 0.1ms longer for every table... not really sure what to do about this (could this be because the database has been globally updated?)

Adding `show_email` to `users` adds negligible overheads (< 0.01ms per user), as this simply appends a column to the in-memory row. This update also does not increase the number of queries to reveal a user record.

7.2.2 Edna Performance Drill-Down

We next break down the cost of Edna's operations into the cost of database operations and the cost of cryptographic operations. [lyt: Each disguise and reveal operation requires decoding of JSON specifications into Edna-specific Rust datatypes; this takes ≈ 0.1ms on average. This could be offloaded to registering each specification with Edna when the application starts, allowing Edna to decode them in advance.] Edna's database operations are fast; in our prototype, they



(a) WebSubmit

(b) HotCRP

Figure 7-2: Applying disguising transformations to previously-decorrelated accounts increases latency linear in the number of pseudoprincipals involved. Hatched lines indicate the proportion of cost attributed to cryptographic operations.

generally take 0.2–0.3ms but vary depending on the amount of data touched. Edna's cryptographic operations are comparatively expensive. PBKDF2 hashing for private key management incurs a 8ms cost and affects account registration and operations on disguised data that reconstruct a user's private key; this accounts for up to 79% of these operations' cost when the operation issues only a few database queries.

Encryption and decryption incur baseline costs of 0.1ms and 0.02ms respectively; their cost grows linearly with data size. In the common case, disguising or revealing data performs two cryptographic operations: one to encrypt/decrypt the diff and speaks-for records, and one to encrypt/decrypt the ID at which they are stored.

Edna also generates a new key for each pseudoprincipal created, which takes 0.2ms. Edna's cryptography accounts for up to 35% of the cost of disguising/revealing operations such as account removal or anonymization; this proportion decreases as the number of database modifications made by a transformation increases. When the application applies multiple disguising transformations and disguises the data of pseudoprincipals, doing so may require several encryptions/decryptions. We evaluate this cost next.

7.2.3 Composing Disguising Transformations.

To understand the overhead of composing transformations in Edna, we measure the cost of composing account removal on top of a prior disguising transformation to anonymize and decorrelate all users' data. We consider WebSubmit and HotCRP, and compare three setups: (*i*) manual account removal (as before); (*ii*) account removal and restoration *without* a prior anonymization disguising transformation; and (*iii*) account removal and restoration *with* a prior anonymization disguising transformation. With prior anonymization, a subset of the user's

data has already been decorrelated when removal occurs, and removal therefore performs per-pseudoprincipal encryptions of disguised data with pseudoprincipals' public keys. Restoring the removed, anonymized account must then individually decrypt pseudoprincipal records and restore them. Hence, disguising and revealing in the third setup should take time proportional to the number of pseudoprincipals created by anonymization.

Figure 7-2 shows the resulting latencies. WebSubmit account removal and restoration latencies increase by ≈1ms per pseudoprincipal (18.2ms and 21.8ms respectively); 50% of this increased cost comes from the additional, per-pseudoprincipal encryption and decryption of records, the rest comes from database operations. HotCRP removal and restoration latencies also increase by ≈1ms per pseudoprincipal (191.2ms and 230.4ms respectively); again, cryptographic operations add ≈0.5ms per pseudoprincipal, and the remaining cost increase comes from per-pseudoprincipal database queries and updates. WebSubmit and HotCRP do not create new references to pseudoprincipals after data gets disguised, but if they did, Edna would need to issue additional per-pseudoprincipal queries to rewrite or remove these references (if configured to do so). Compared to accounts in WebSubmit, accounts in HotCRP have more data and 14–15× more pseudoprincipals after anonymization, which accounts for the larger relative slowdown.

Importantly, disguising latencies stabilize when Edna composes further disguising transformations: since cost is proportional to the number of pseudoprincipals affected, latency does not grow once the application has maximally decorrelated data (to one pseudoprincipal per record), as done by HotCRP anonymization.

7.3 Edna Overheads

Edna adds both space and compute overheads to the application; we measure the impact of these next.

7.3.1 Space Used By Edna.

To understand Edna's space footprint, we measure the size of all data stored on disk by Edna before and after 10% of users in Lobsters (1.6k users) remove their accounts. Cryptographic material adds overhead and each generated pseudoprincipal adds an additional user to the application database; Edna also stores data for each registered principal (a public key and a list of opaque indexes) as well as encrypted records.

Edna's record storage uses 12 MB, which grows to 58.5 MB after the users remove their accounts, and the application database size increases from 261 MB to 290 MB (+11%). (Edna also caches some of this data in memory.) The space used is primarily proportional to the number of pseudoprincipals produced: each pseudoprincipal requires storing an application database record, a speaks-for record, and row in the principal table. In this experiment, Lobsters produces 78.1k pseudoprincipals. Edna removes the public keys and database data for the 1.6k

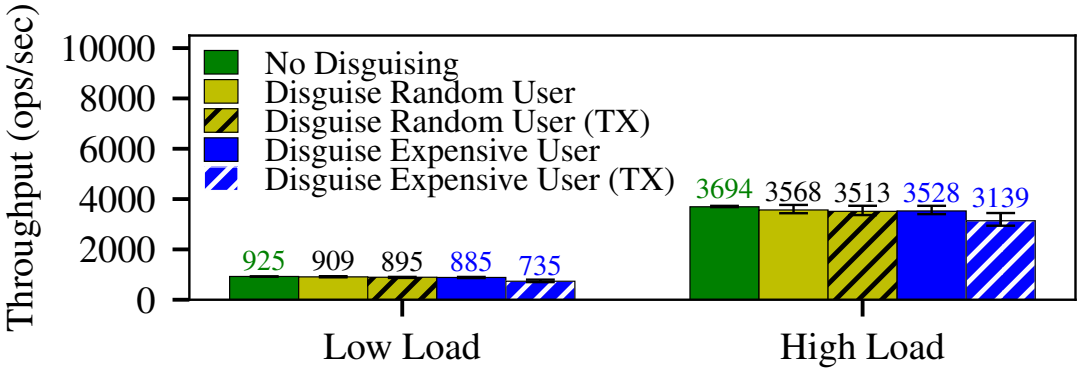


Figure 7-3: Continuous disguising/revealing operations in Lobsters have a <7% impact on application request throughput when disguising a random user; an extreme case of a heavy-hitter user with lots of data repeatedly disguising and revealing causes a 3–17% drop in throughput.

removed principals, but stores encrypted diff records with their information, which uses another 2.2 MB.

7.3.2 Impact On Concurrent Application Use.

For Edna to be practical, the throughput and latency of normal application requests by other users must be largely unaffected by Edna's disguising and revealing operations.

We thus measure the impact of Edna's operations on other concurrent requests in Lobsters. In the experiment, a set of users make continuous requests to the application that simulate normal use, while another distinct set of users continuously remove and restore their accounts. Edna applies disguising transformations sequentially, so only one transformation happens at a time. We measure the throughput of "normal" users' application operations, both without Edna operations (the baseline) and with the application continuously invoking Edna. The Lobsters workload is based on request distributions in the real Lobsters deployment [32].

Since users' disguising/revealing costs vary in Lobsters, we measure the impact of (*i*) randomly chosen users invoking account removal/restoration, and (*ii*) the user with the most data continuously removing and restoring their account (a worst-case scenario). We show throughput in a low load scenario (≈20% CPU load), and a high load scenario (≈95% CPU load). Finally, we measure settings with and without a transaction for Edna transformations. A good result for Edna would show little impact on normal operation throughput when concurrent disguising transformations occur.

Figure 7-3 shows the results. If a random user disguises and reveals their data (the common case), normal operations are mostly unaffected by concurrent disguising and revealing: throughput drops ≤3.7% without transactions and ≤7.0% with transactions. Constantly disguising and revealing the user with the most data (the worst-case scenario) has a larger effect, with throughput reduced by up to 7.4% (without transactions) and up to 17% (with transac-

tions, high load).

This shows that Edna's disguising and revealing transformations have acceptable impact on other users' application experience in the common case.

The latency of disguising operations depends on load: the expensive user's account removal and revealing take 4.4 and 3.6 seconds under high load, and 3.3 and 2.6 seconds under low load. This is acceptable: 50% of data deletions at Facebook take five minutes or longer to complete [17].

7.4 Comparison to Qapla

We compare Edna's performance and the effort to use Edna to an implementation of the same disguising and revealing functionality for WebSubmit in Qapla.

Effort. Specifying disguising transformations as Qapla policies requires far more explicit reasoning about transformations' implementations and their compositions. In Qapla, a developer would realize disguising transformations via metadata flags that they add to the schema (e.g., `is_deleted` for removed data) and toggles in application code. They then provision Qapla with a predicate that checks if this metadata flag is `true` before returning a row. Developers must carefully craft Qapla's predicates, which grow in complexity with the number of disguising transformations that can compose. For example, an application supporting both account removal and account anonymization must combine predicates such that removal always takes precedence. Each additional transformation increases the number of predicates whose combinations the developer must reason about. Developers must also optimize Qapla predicates (e.g., reducing joins, adding schema indexes and index hints) to achieve reasonable performance.

To modify data, the application developer can use Qapla's "cell blinding" mode, which dynamically changes column values (to fixed values) based on a predicate before returning query results. The developer must manually implement more complex modifications and decorrelation (i.e., creating pseudoprincipals and rewriting foreign keys).

Realizing WebSubmit transformations in Qapla required 576 lines of C/C++, and 110 lines of Rust to add pseudoprincipal, modification, and decorrelation support.

Overall, Qapla requires more developer effort than Edna, particularly in writing composable and performant predicates, and manually implementing modifications and decorrelations. However, Qapla's approach does make some things easier. Because data remains in the database, revealing simply requires toggling metadata flags, and data to reveal can adapt to database changes (e.g., schema updates). But keeping the data in the database also means that developers cannot use Qapla to achieve GDPR-compliant data removal.

Performance. We measure Qapla's performance (Figure 7-4) on the same WebSubmit operations (Figure 7-1a). Qapla performs well on operations that require only writes, since Qapla does not rewrite write queries. Removing and restoring accounts requires only a single metadata flag update in Qapla, whereas Edna encrypts/decrypts user data and actually deletes it from the

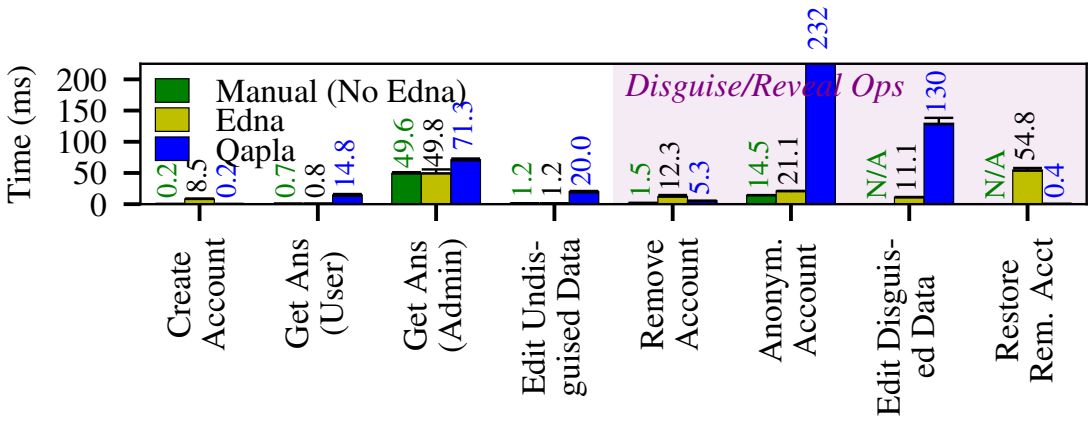


Figure 7-4: Edna achieves competitive performance with a manual baseline and outperforms Qapla on nearly all common WebSubmit operations (2k users, 80 answers/user). Bars show medians, error bars are 5^{th}/95^{th} percentile latencies.

database. However, Qapla rewrites all read queries, so Qapla performs poorly on operations that require reads, such as listing answers and editing (disguised or undisguised) data. Qapla's query rewriting takes ≈1ms, and rewrites `SELECT` queries in ways that affect performance (e.g., adding joins to evaluate predicates). Overall, Edna achieves better performance on common operations.

7.5 Edna+CryptDB

We combine Edna with CryptDB to evaluate the cost of composing Edna's guarantees with those of encrypted databases. CryptDB protects undisguised database contents against attackers who compromise the database server itself (with some limitations [31]), in addition to Edna's existing protections for disguised data.

Edna+CryptDB operates in CryptDB's threat model 2 (database server and proxy can be compromised). A developer using Edna+CryptDB deploys the application (and Edna) atop a proxy that encrypts and decrypts database rows. Queries from Edna and the application operate unchanged atop the proxy, but to ensure proper access to user data, the application and Edna must handle user sessions. Edna+CryptDB exposes an API to log users in and out using their credentials. Prior to applying transformations to a user's data, Edna performs a login to ensure that Edna has legitimate access to their data (e.g., the user, an admin, or someone sharing the data is logged in).

Edna+CryptDB handles keys in the same way as CryptDB: Edna+CryptDB encrypts database rows with per-object keys, and object keys are themselves encrypted with the public keys of the users who can access the object. After a user logs in, the application gives the proxy their private

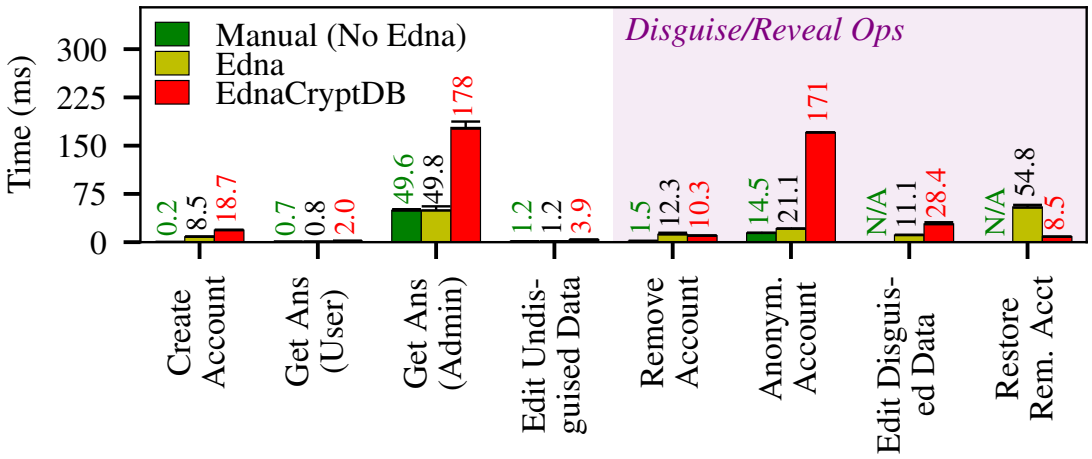


Figure 7-5: Latencies of WebSubmit (2k users, 80 answers/user) operations when implemented with Edna+CryptDB (adding encrypted database support). Bars show median latency; error bars are $5^{th}/95^{th}$ percentile latencies.

key, thus allowing decryption of their accessible objects.

Our prototype only supports the CryptDB deterministic encryption scheme (AES-CMC encryption), which limits it to equality comparison predicates. It also does not support joins, a limitation shared with multi-principal CryptDB.

Performance. We measure the latency of WebSubmit operations like before, and compare a manual baseline, Edna, and Edna+CryptDB. Edna+CryptDB is necessarily more expensive than Edna, and a good result for Edna+CryptDB would therefore show moderate overheads over Edna, and acceptable absolute latencies.

Figure 7-5 shows the results. Normal application operations are 2–3× slower with Edna+CryptDB than in Edna, with the largest overheads on operations that access many rows, such as the admin viewing all answers. Disguising and revealing operations are also 2–7× slower than Edna.

These overheads result from the cryptographic operations and additional indirection in Edna+CryptDB. Edna+CryptDB relies on a MySQL proxy, which adds latency: a no-op version of our proxy makes operations 1.03-1.5× slower. Cryptographic operations themselves are cheap (< 0.2ms), but every object inserted, updated, or read also requires lookups to find out which keys to use, query rewriting to fetch the right encrypted rows, and execution of more complex queries.

This is particularly expensive when the user owns many keys (e.g., the WebSubmit admin). Admin-applied anonymization incurs the highest overhead (+156.4ms) as it issues many queries to read user data and execute decorrelations. Among the common operations, an admin getting all the answers for a lecture suffers similar overheads (+127.7ms).

Like CryptDB, Edna+CryptDB increases the database size (4–5× for our WebSubmit proto-

type). Edna+CryptDB also stores an encrypted object key and its metadata (1KB per key) for each user with access to that object.

7.6 Support for Application Updates and Schema Migrations

In addition to the updates described and implemented in §6.1, we inspected 20 other updates/migrations in Lobsters from the past three years, and classified them broadly as follows:

- creating tables;
- removing, adding, or renaming table columns;
- removing or adding table indexes;
- setting column constraints (e.g., `NOT NULL`);
- modifying column content based on deterministic functions (e.g., URL normalization);
- generating rows for new tables based on existing table rows (e.g., adding `story_texts` when restructuring `stories`); and
- updating a row based on the current state of other database tables (e.g., updating the `vote_count` of comments based on the number of votes).

Edna can support all but the last category of updates, unless the update's output is correct no matter the state of other database tables. This last category represents 2 of the 20 inspected updates. We note that these types of updates occur in applications that optimize for performance by denormalizing their database schema; we hypothesize that this category of update would not exist with a normalized schema. These particular updates also happen to be idempotent for any rows affected, and thus could be reapplied via e.g., a cron job regularly to update revealed data.

With the exception of this last category of updates, users can still reveal their data disguised prior to these migrations, as if the migration had occurred with their data present in the database.

7.7 Summary

In our evaluation, we used Edna to add seven disguising transformations to three web applications. We found that the effort required was reasonable, that Edna's disguising and revealing operations are fast enough to be practical, and that they impose little overhead on normal application operation.

Chapter 8

Conclusion

We increasingly live in a world where user data lies in the hands of web services, removing users' ability to control the privacy of their data. This thesis presents a possible way forward to give users back flexible privacy controls over their data, envisioning a world where web services routinely protect, store, and reveal disguised user data with user permission.check With Edna, developers can provide data disguising and revealing transformations that give users control over their data in web applications. Throughout this thesis, we describe how these transformations help users protect inactive accounts, selectively dissociate personal data from public profiles, and remove a web service's access to their data without permanently losing their accounts. We showed that Edna supports disguised data in real-world applications with reasonable performance.

However, disguised data and flexible user privacy more broadly encounter several challenges that Edna leaves unsolved, which we describe here as future opportunities to improve the state of user data privacy in web applications.

8.1 Edna in the Real World

While Edna enables disguised data in many web applications, supporting disguised data in all web applications—particularly those at scale—raises several open questions, which we discuss here.

Disguised Data Storage. If e.g., millions of users disguise their data over time, disguised data may become too much for applications to store. Even though storage is historically cheap, and applications today willingly store and retain user data nearly indefinitely, eventually disguised data (particularly since it cannot be sold or processed) may burden the application. Edna currently retains disguised data in its disguised table until a user reveals their disguised data, which could be forever if users choose to never reveal data. Instead, Edna could allow applications to put reasonable, coarse-grained time limits (e.g., 10 years) on disguised data to eventually clean

it up, without leaking fine-grained information about which data was disguised at the same time.

Deriving Value from Disguised Data. As mentioned, even if applications *do* store disguised data, the cost of doing so may not pay off, since applications cannot derive value from it because it remains confidential. We believe that with clever cryptography such as additive homomorphic secret sharing as done in Zeph [10], Edna could potentially support limited computation such as aggregations over disguised data from which an application can derive value. However, such cryptographic approaches require some initial setup to determine whose data can be aggregated with whose (in order to split the secrets appropriately) and who should be able to reveal the result (can individual users also perform aggregates, or just the application?) Futhermore, defining groups of users whose data can be combined requires more thought into what these groups may reveal about their constituents. Homomorphic encryption techniques are also historically regarded as impractical due to their performance overheads.

Edna in a Distributed System Distributed Edna, parallel Edna, what else is needed to make Edna real?

8.2 Disguising Beyond Edna: Flexible Access Control

Zooming out beyond disguised user data for web applications, we can imagine completely different directions in which disguising for flexible privacy could be applied. We describe here using many of the same disguising abstractions to provide flexible access control, where the application itself needs to control user data exposure to insiders.

Today, thousands of insiders at companies have access to sensitive user data, and this access can lead to devastating data leaks. cite? One important part of this problem stems from the fact that today, employees have access to all the data that they need for their job at any point in time. For example, a hotel employee might be given authorization to see the full room booking and customer database because they need this information to perform their job, such as performing aggregates to determine room occupancy; or helping customers manage their bookings However, we make the key observation that these employees do not *always* need *all* of this information.

Instead, we envision a world where every employee operates atop a personalized redacted database that contains the minimum amount of data they need continuously. For example, a hotel customer service representative might only need to be able to look at room occupancy and see free rooms when not actively helping a customer. Similarly, a business intelligence analyst might need to be able to compute aggregates and statistics using coarse-grained data like area codes, but not need to see unique user identifiers like usernames or precise phone numbers.

However, redacted databases would present only the minimal amount of data the employee

needs for common, daily operations. If, for example, a particular customer needs assistance, the employee currently helping the customer should be able to change their redactions temporarily and dynamically to expose only the necessary parts of that user's data. Different employees simultaneously helping different customers can apply different upgrades to get a partially redacted view of the database individualized for that employee. As soon as the employee no longer needs the user's data, the employee's access returns back to its original, minimally exposing state.

To achieve this world of flexible, minimalist access control, we need a database system that can dynamically change redactions that apply to each employee, but present a consistent view of the redacted database to each employee. A strawman solution might attempt to do so with personalized database views (similar to the proposal in Multiverse DB [42]), but this would require an enormous number of views (potentially one for each employee, and for each upgrade the employee applies). Furthermore, redactions would need to handle referential integrity, work even in the presence of indexes and database views, and work with existing query optimizations without overly affecting performance. Employees must also have the ability to flexibly perform fine-grained upgrades of their access without compromising security. While disguising and revealing abstractions can help address some of these challenges (e.g., handling referential integrity when performing redactions, and providing ways to specify redactions and upgrades), achieving a database system for flexible access control requires solving many new challenges.

Bibliography

[1] Lobsters. URL https://lobste.rs.

[2] Archita Agarwal, Marilyn George, Aaron Jeyaraj, and Malte Schwarzkopf. Retrofitting gdpr compliance onto legacy databases. *Proc. VLDB Endow.*, 15(4):958–970, dec 2021. ISSN 2150-8097. doi: 10.14778/3503585.3503603. URL https://doi.org/10.14778/3503585.3503603.

[3] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. Blockstack: A global naming and storage system secured by blockchains. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 181–194, Denver, Colorado, USA, June 2016. ISBN 978-1-931971-30-0. URL https://www.usenix.org/conference/atc16/technical-sessions/presentation/ali.

[4] Apple. Apple platform security, April 2023. URL https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf.

[5] Daniel Arkin. Celebrities are starting to leave twitter. here's a running list. URL https://www.nbcnews.com/pop-culture/celebrity/twitter-celebrities-leaving-elon-musk-rcna54831.

[6] Daniel Augus. Thinking of quitting twitter? here's the right way to do it. URL https://economictimes.indiatimes.com/news/how-to/thinking-of-quitting-twitter-heres-the-right-way-to-do-it/articleshow/95696874.cms.

[7] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 566–582, November 2001.

[8] T. Berners-Lee, R. Fielding, and L. Masinter. Rfc2396: Uniform resource identifiers (uri): Generic syntax, 1998.

[9] Kristy Browder and Mary Ann Davidson. The virtual private database in oracle9ir2. *Oracle Technical White Paper, Oracle Corporation*, 500(280), 2002.

[10] Lukas Burkhalter, Nicolas Küchler, Alexander Viand, Hossein Shafagh, and Anwar Hithnawi. Zeph: Cryptographic enforcement of end-to-end data privacy. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 387–404, July 2021. ISBN 978-1-939133-22-9. URL https://www.usenix.org/conference/osdi21/presentation/burkhalter.

[11] California Legislature. The California Consumer Privacy Act of 2018, June 2018. URL https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375.

[12] Michelle Caruthers. World password day: How to improve your passwords, April 2023. URL https://blog.dashlane.com/world-password-day/.

[13] Tej Chajed, Jon Gjengset, Jelle van den Hooff, M. Frans Kaashoek, James Mickens, Robert Morris, and Nickolai Zeldovich. Amber: Decoupling user data from web applications. In *Proceedings of the 15th Workshop on Hot Topics in Operating Systems (HotOS)*, Kartause Ittingen, Switzerland, May 2015. URL https://www.usenix.org/conference/hotos15/workshop-program/presentation/chajed.

[14] Tej Chajed, Jon Gjengset, M. Frans Kaashoek, James Mickens, Robert Morris, and Nickolai Zeldovich. Oort: User-centric cloud storage with global queries. Technical Report MIT-CSAIL-TR-2016-015, MIT CSAIL, 2016. URL https://people.csail.mit.edu/nickolai/papers/chajed-oort.pdf.

[15] Ramesh Chandra, Priya Gupta, and Nickolai Zeldovich. Separating web applications from user data storage with bstore. In *Proceedings of the 2010 USENIX Conference on Web Application Development (WebApps)*, USA, 2010.

[16] Adam Chlipala. Static checking of dynamically-vary-ing security policies in database-backed applications. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 105–118, 01 2010.

[17] Katriel Cohn-Gordon, Georgios Damaskinos, Divino Neto, Joshi Cordova, Benoít Reitz, Benjamin Strahs, Daniel Obenshain, Paul Pearce, and Ioannis Papagiannis. DELF: Safeguarding deletion correctness in online social networks. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security)*, August 2020. URL https://www.usenix.org/conference/usenixsecurity20/presentation/cohn-gordon.

[18] Kate Conger and Lauren Hirsch. Elon musk completes $44 billion deal to own twitter. URL https://www.nytimes.com/2022/10/27/technology/elon-musk-twitter-deal-complete.html.

[19] Kinan Dak Albab, Ishan Sharma, Justus Adam, Benjamin Kilimnik, Aaron Jeyaraj, Raj Paul, Artem Agvanian, Leonhard Spiegelberg, and Malte Schwarzkopf. K9db: Privacy-compliant storage for web applications by construction. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, July 2023.

[20] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: An encrypted search system with distributed trust. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1101–1119, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/dauterman-dory.

[21] Emma Dauterman, Vivian Fang, Ioannis Demertzis, Natacha Crooks, and Raluca Ada Popa. Snoopy: Surpassing the scalability bottleneck of oblivious storage. In *Proceedings of the 28th ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, page 655–671, 2021. ISBN 9781450387095. URL https://doi.org/10.1145/3477132.3483562.

[22] Vincent De Beer. Heartbreak and hacking: Dating apps in the pandemic, April 2021. URL https://securityboulevard.com/2021/04/heartbreak-and-hacking-dating-apps-in-the-pandemic/.

[23] Amol Deshpande. Sypse: Privacy-first data management through pseudonymization and partitioning. In *Proceedings of the 11th Conference on Innovative Data Systems Research (CIDR)*, Chaminade, California, USA, January 2021.

[24] Katie Doptis. Podcast: How AWS KMS could help customers meet encryption and deletion requirements, including GDPR, June 2018. URL https://aws.amazon.com/blogs/security/podcast-how-aws-kms-could-help-customers-meet-encryption-and-deletion-requirements-inc

[25] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdh aswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019. URL https://www.flux.utah.edu/paper/duplyakin-atc19.

[26] European Union. 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*, L119:1–88, May 2016. URL http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC.

[27] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, page 657–666, New York, NY, USA, 2007. ISBN 9781595936547. doi: 10.1145/1242572.1242661. URL https://doi.org/10.1145/1242572.1242661.

[28] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proceedings of the 18th USENIX Security Symposium (USENIX Security)*, pages 299–316, 2009. URL http://dl.acm.org/citation.cfm?id=1855768.1855787.

[29] Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Maziéres, John C. Mitchell, and Alejandro Russo. Hails: Protecting data privacy in untrusted web applications. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 47–60, Hollywood, California, USA, October 2012. ISBN 978-1-931971-96-6. URL https://www.usenix.org/conference/osdi12/technical-sessions/presentation/giffin.

[30] GlobalData Thematic Research. Crypto-shredding the best solution for cloud system data erasure, January 2022. URL https://www.verdict.co.uk/crypto-shredding-gdpr-cloud-systems/.

[31] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. Why your encrypted database is not secure. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS)*, page 162–168, 2017. ISBN 9781450350686. doi: 10.1145/3102980.3103007. URL https://doi.org/10.1145/3102980.3103007.

[32] Peter Bhat Harkins. Lobste.rs access pattern statistics for research purposes, March 2018. URL https://lobste.rs/s/cqnzl5/lobste_rs_access_pattern_statistics_for#c_hj0r1b.

[33] Katia Hayati and Martín Abadi. Language-based enforcement of privacy policies. In *Proceedings of the International Workshop on Privacy Enhancing Technologies*, pages 302–313. Springer, 2004.

[34] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 533–549, 2016. ISBN 978-1-931971-33-1. URL http://dl.acm.org/citation.cfm?id=3026877.3026919.

[35] Zsolt István, Soujanya Ponnapalli, and Vijay Chidambaram. Software-defined data protection: low overhead policy compliance at the storage layer is within reach! *Proc. VLDB Endow.*, 14(7):1167–1174, mar 2021. ISSN 2150-8097. doi: 10.14778/3450980.3450986. URL https://doi.org/10.14778/3450980.3450986.

[36] Burt Kaliski. Pkcs# 5: Password-based cryptography specification version 2.0. Technical report, 2000.

[37] Eddie Kohler. Hotcrp.com. URL https://hotcrp.com.

[38] Eddie Kohler. Hotcrp.com privacy policy. HotCRP.com, August 2020. URL https://hotcrp.com/privacy.

[39] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. Pesos: policy enhanced secure object store. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355841. doi: 10.1145/3190508.3190518. URL https://doi.org/10.1145/3190508.3190518.

[40] Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner. Schengendb: A data protection database proposal. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 24–38. Springer, 2019. ISBN 978-3-030-33752-0.

[41] Maxwell Krohn, Alex Yip, Micah Brodsky, Robert Morris, and Michael Walfish. A World Wide Web Without Walls. In *Proceedings of the 6th ACM Workshop on Hot Topics in Networking (HotNets)*, Atlanta, Georgia, USA, November 2007.

[42] Alana Marzoev, Lara Timbó Araújo, Malte Schwarzkopf, Samyukta Yagati, Eddie Kohler, Robert Morris, M. Frans Kaashoek, and Sam Madden. Towards multiverse databases. In *Proceedings of the 17th Workshop on Hot Topics in Operating Systems (HotOS)*, pages 88–95, 2019.

[43] Mastadon gGmbH. Social networking that's not for sale., 2023. URL https://joinmastodon.org/.

[44] Cecily Mauran. So you want to leave twitter: A wellness plan. URL https://mashable.com/article/how-to-leave-twitter-guide-elon-musk.

[45] Aastha Mehta, Eslam Elnikety, Katura Harvey, Deepak Garg, and Peter Druschel. Qapla: Policy compliance for database-backed systems. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, pages 1463–1479, Vancouver, British Columbia, August 2017. ISBN 978-1-931971-40-9. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/mehta.

[46] Richard Mortier, Jianxin Zhao, Jon Crowcroft, Liang Wang, Qi Li, Hamed Haddadi, Yousef Amar, Andy Crabtree, James Colley, Tom Lodge, et al. Personal data management with the databox: What's inside the box? In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, pages 49–54, 2016.

[47] Mozilla Foundation. Tinder (*privacy not included), March 2021. URL https://foundation.mozilla.org/en/privacynotincluded/tinder/.

[48] Ken Olin. Hey all - i'm out of here. no judgement... URL https://twitter.com/kenolin1/status/1586031904007954433.

[49] Shoumik Palkar and Matei Zaharia. Diy hosting for online privacy. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets)*, pages 1–7, 2017.

[50] Primal Pappachan, Roberto Yus, Sharad Mehrotra, and Johann-Christoph Freytag. Sieve: A middleware approach to scalable access control for database management systems. *arXiv preprint arXiv:2004.07498*, 2020.

[51] Pranay Parab. How to make a burner account on reddit, even though they don't want you to anymore. Lifehacker, January 2022. URL https://lifehacker.com/how-to-make-a-burner-account-on-reddit-even-though-the-1848336857.

[52] Nicole Perlroth, Amie Tsang, and Addam Satariano. Marriott hacking exposes data of up to 500 million guests, November 2018. URL https://www.nytimes.com/2018/11/30/business/marriott-data-breach.html.

[53] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23th ACM Symposium on Operating Systems Principles (SOSP)*, page 85–100, 2011. ISBN 9781450309776. doi: 10.1145/2043556.2043566. URL https://doi.org/10.1145/2043556.2043566.

[54] Raluca Ada Popa, Emily Stark, Jonas Helfer, Steven Valdez, Nickolai Zeldovich, M. Frans Kaashoek, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 157–172, 2014. ISBN 978-1-931971-09-6. URL http://dl.acm.org/citation.cfm?id=2616448.2616464.

[55] Joel Reardon, Srdjan Capkun, and David Basin. Data node encrypted file system: Efficient secure deletion for flash memory. In *Proceedings of the 21st USENIX Security Symposium (USENIX Security)*, pages 333–348, Bellevue, Washington, USA, August 2012. ISBN 978-1-931971-95-9. URL https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/reardon.

[56] Shonda Rhimes. Not hanging around for whatever elon has planned. bye. URL https://twitter.com/shondarhimes/status/1586399694896390147.

[57] Ruby on Rails Guides. Active Record Migrations, 2024. URL https://guides.rubyonrails.org/active_record_migrations.html.

[58] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. Solid: a platform for decentralized social applications based on linked data. Technical report, MIT CSAIL & Qatar Computing Research Institute, 2016.

[59] David Schultz and Barbara Liskov. Ifdb: Decentralized information flow control for databases. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*, pages 43–56, April 2013.

[60] Malte Schwarzkopf. websubmit-rs: a simple class submission system. URL https://github.com/ms705/websubmit-rs.

[61] Malte Schwarzkopf, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. Position: Gdpr compliance by construction. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 39–53, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33752-0.

[62] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979. ISSN 0001-0782. doi: 10.1145/359168.359176. URL https://doi.org/10.1145/359168.359176.

[63] Elizabeth Stobert and Robert Biddle. The password life cycle: User behaviour in managing passwords. In *Proceedings of the 10thUSENIX Conference on Usable Privacy and Security (SOUPS)*, page 243–255, 2014. ISBN 9781931971133.

[64] Yang Tang, Phillip Ames, Sravan Bhamidipati, Ashish Bijlani, Roxana Geambasu, and Nikhil Sarda. CleanOS: Limiting mobile data exposure with idle eviction. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 77–91, Hollywood, California, USA, October 2012. ISBN 978-1-931971-96-6. URL https://www.usenix.org/conference/osdi12/technical-sessions/presentation/tang.

[65] The Office of the Privacy Commissioner. New zealand's biggest data breach shows retention is the sleeping giant of data security, April 2023. URL https://www.privacy.org.nz/publications/statements-media-releases/new-zealands-biggest-data-breach-shows-retention-is-the-sleeping-giant-of-data-securit

[66] Patrick Townsend. GDPR, Right of Erasure (Right to be Forgotten), and Encryption Key Management, January 2028. URL https://info.townsendsecurity.com/gdpr-right-erasure-encryption-key-management.

[67] Harshavardhan Unnibhavi, David Cerdeira, Antonio Barbalace, Nuno Santos, and Pramod Bhatotia. Secure and policy-compliant query processing on heterogeneous computational storage architectures. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1462–1477, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517913. URL https://doi.org/10.1145/3514221.3517913.

[68] U.S. Department of Education. Ferpa, August 1974. URL https://studentprivacy.ed.gov/ferpa.

[69] Frank Wang, Ronny Ko, and James Mickens. Riverbed: Enforcing user-defined privacy constraints in distributed web services. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 615–630, Boston, Massachusetts, USA, February 2019. ISBN 978-1-931971-49-2. URL https://www.usenix.org/conference/nsdi19/presentation/wang-frank.

[70] Caity Weaver and Danya Issawi. 'finsta,' explained. The New York Times, September 2021. URL https://www.nytimes.com/2021/09/30/style/finsta-instagram-accounts-senate.html.

[71] Scott Wolchok, Owen S. Hofmann, Nadia Heninger, Edward W. Felten, J. Alex Halderman, Christopher J. Rossbach, Brent Waters, and Emmett Witchel. Defeating vanish with low-cost sybil attacks against large dhts. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, California, USA, February 2010. URL https://www.ndss-symposium.org/ndss2010/defeating-vanish-low-cost-sybil-attacks-against-large-dhts.

[72] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. *ACM SIGPLAN Notices*, 47(1):85–96, 2012.

[73] Frances F Yao and Yiqun Lisa Yin. Design and analysis of password-based key derivation functions. In *Topics in Cryptology (CT-RSA): The Cryptographers' Track at the RSA Conference*, pages 245–261, San Francisco, California, USA, 2005. Springer.

[74] Wen Zhang, Eric Sheng, Michael Chang, Aurojit Panda, Mooly Sagiv, and Scott Shenker. Blockaid: Data access policy enforcement for web applications. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 701–718, Carlsbad, California, USA, July 2022. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/zhang.