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

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

Users' privacy requirements for their sensitive data on web applications are highly contextual and constantly in flux. 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

Thanks

Contents

List of Figures

List of TODOs

Chapter 1

Introduction

Users' privacy requirements for their sensitive data on web applications are highly contextual and constantly in flux. 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.

1.1 Motivation

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 [**? ? ?**]. And while users now have the right to delete their data (via e.g., the GDPR [**?**] or CCPA [**?**]), users want and deserve more nuanced controls over their data that don't exist today.

Consider Twitter: after a change in management [**?**], many users wanted to leave the platform and try out alternatives (e.g., Mastodon). But each user faced a tricky question: should they keep their Twitter account, or should they delete it? Advice on how to quit Twitter [**? ?**] 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. Hence, many users left Twitter but kept their accounts [**? ? ?**]. 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 [**?** **?**] 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 [**?** **?**]. 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 [**?**] and Instagram "finstas" [**?**]), 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. This thesis introduces the new abstraction of *disguised data* that enables this functionality and supports these use cases.

1.2 Why is this hard?

Today, 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. Furthermore, stored data must be inaccessible to the application and protected against data breaches, but must be accessible if the user chooses to return.

Developers would also have to reason about interactions between multiple data-redacting features. 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.

1.3 Our Approach

To tackle the problem of flexible privacy in web applications, we present a system design that moves 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. We capture this goal with 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.

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.

1.4 Challenges

Edna's approach faces three challenges. 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

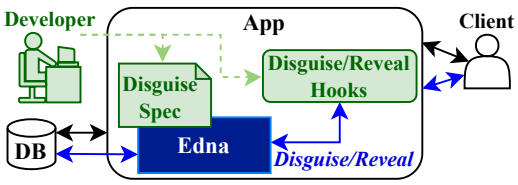


Figure 1-1: Developers write disguise specifications and add hooks to invoke Edna from the application (green); in normal operation, clients use these hooks in the application to disguise and reveal their data in the database (blue).

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, and unlocks and restores data to the service only when a user provides their credentials to reveal.

1.5 Edna Overview

Edna helps developers realize new options for users to control their data via *disguising 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 **??**). 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.

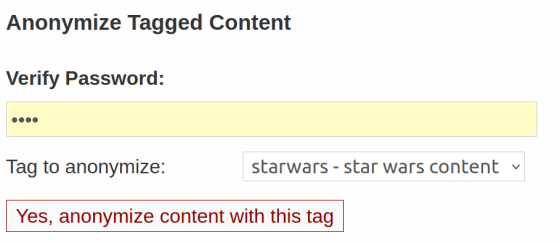
(3) When a user wishes to reveal their disguised data, they pass credentials to the application, which calls into Edna to reveal the data. Credentials are application-specific: users may either provide their private key or other credentials sufficient for Edna to re-derive the private key. Edna reads 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).

```
// 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 1-2: Lobsters topic-based anonymization disguise specification (JSON pseudocode), which decorrelates comments and removes votes on stories with the specific topic tag.

1.6 Example: Lobsters Topic Anonymization




```
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 1-3: The Lobsters developer adds a hook in the UI and code to perform topic-based anonymization.

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

1.7 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 [**? ? ? ?**].

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.

1.8 Contributions

This thesis makes the following contributions:

1. The identification and exploration of the problem of *flexible privacy* for user data in web applications.
2. Abstractions to achieve flexible privacy in web applications via *disguised data*, including abstractions for disguising and revealing, and a small set of data-anonymizing primitives (remove, modify, decorrelate) that cover a wide range of application needs and compose cleanly.
3. Techniques to implement these abstractions, including pseudoprincipals, speaks-for and diff records, and speaks-for chains.
4. A design and realization of these techniques in Edna, a prototype library that implements user data control via disguising and revealing.
5. Case studies that integrate Edna with three real-world web applications and demonstrate Edna's ability to enable composable and reversible transformations.
6. An evaluation of Edna's effectiveness and performance, including how Edna contrasts with and complements related work (Qapla [**?**] and CryptDB [**?**]).

1.9 Remaining Work: Revealing with Schema Migrations and Application Updates

In Edna's current design, **application updates** that implicitly enforce invariants on application data remain unknown (and thus uncheckable) by Edna. Edna also fails to reveal disguised data affected by **schema migrations** performed since the time of disguise. We will add an API for developers to log important updates and schema migrations with Edna, which Edna will apply to disguised data prior to restoring it to the database.

Edna will apply these updates or schema migrations prior to performing Edna's existing consistency checks; these checks enable Edna to detect if revealing transformations will violate referential integrity or other structual database invariants (e.g., uniqueness requirements).

Chapter 2

Related Work

Disguised data provides flexible (and reversible) degrees of data privacy. 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.

Data deletion tools, such as DELF at Meta [**?**] and K9DB [**?**] help correctly delete data. While disguised data states can provide GDPR-compliant account deletion, disguised data 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.

Policy enforcement systems such as Qapla [**?**], Blockaid [**?**] and others [**? ? ? ? ? ? ? ? ?**] aim to prevent unauthorized access to data and protect against leakage via compromised accounts or SQL injections. However, policy-enforcing systems do not help users anonymize data or maintain application integrity constraints, which is the explicit goal of disguised data. Database contents of data under disguise is also modified so sensitive data (the data under disguise) 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.

Encrypted storage systems such as CryptDB [**?**] and Mylar [**?**] protect against database server compromise, with some limitations [**?**]. 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.

Other related work uses crytographic or systems security methods to prove enforcement of user-defined policies [**? ?**], or explores new decentralized paradigms for web application operation [**? ? ? ? ? ? ? ?**]. These may restrict application functionality, whereas a system for disguised data can leave the data and business models unchanged.

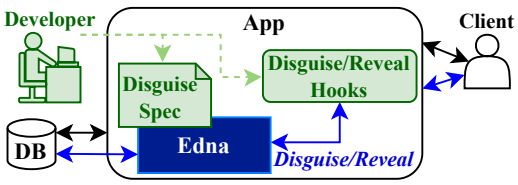


Figure 3-1: Developers write disguise specifications and add hooks to invoke Edna from the application (green); in normal operation, clients use these hooks in the application to disguise and reveal their data in the database (blue).

Chapter 3

Overview

3.0.1 Edna Overview

Edna helps developers realize new options for users to control their data via *disguising 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 **??**). 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.

(3) 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

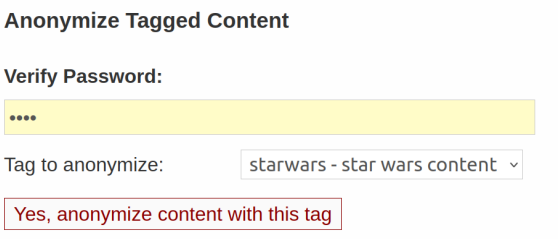
```
// Decorrelate comments on stories w/tag {{TAG}}
"comments": [{
  "type": "Decorrelate",
  "predicate": "tags.tag = {{TAG}}",
  "from": "comments JOIN stories ON comments.story_id = stories.id
           JOIN taggings ON stories.id = taggings.story_id
           JOIN tags ON...",
  "group_by": "stories.id",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

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).

3.0.2 Example: Lobsters Topic Anonymization




```
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.

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

3.0.3 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 [**? ? ? ?**].

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

Design

This thesis makes the following contributions:

1. The identification and exploration of the problem of *flexible privacy* for user data in web applications.
2. Abstractions to achieve flexible privacy in web applications via *disguised data*, including abstractions for disguising and revealing, and a small set of data-anonymizing primitives (remove, modify, decorrelate) that cover a wide range of application needs and compose cleanly.
3. Techniques to implement these abstractions, including pseudoprincipals, speaks-for and diff records, and speaks-for chains.
4. A design and realization of these techniques in Edna, a prototype library that implements user data control via disguising and revealing.
5. Case studies that integrate Edna with three real-world web applications and demonstrate Edna's ability to enable composable and reversible transformations.
6. An evaluation of Edna's effectiveness and performance, including how Edna contrasts with and complements related work (Qapla [**?**] and CryptDB [**?**]).

Appended to this proposal is the Edna paper describing Edna's design, implementation, and evaluation.

4.0.1 Remaining Work: Revealing with Schema Migrations and Application Updates

In Edna's current design, **application updates** that implicitly enforce invariants on application data remain unknown (and thus uncheckable) by Edna. Edna also fails to reveal disguised data affected by **schema migrations** performed since the time of disguise. We will add an API for developers to log important updates and schema migrations with Edna, which Edna will apply to disguised data prior to restoring it to the database.

Edna will apply these updates or schema migrations prior to performing Edna's existing consistency checks; these checks enable Edna to detect if revealing transformations will violate

referential integrity or other structual database invariants (e.g., uniqueness requirements).

Chapter 5

Implementation

Implementation

Chapter 6

Evaluation

6.1 Case Studies

This section evaluates Edna by using it to add new data-redacting features to several applications; §**??** evaluates the effort needed to do so and the resulting performance.

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

6.1.1 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 §**??**, 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.

6.1.2 WebSubmit

We integrated Edna as a Rust library with WebSubmit [**?**]. 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 [**?**] 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.

6.1.3 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 [**?**]. HotCRP currently retains past conference data forever and requires manual requests for account removal [**?**]. 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.

Our evaluation seeks to answer five questions:

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

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 [**?**] using Qapla's query rewriting and access control policies; and (*iii*) an integration of Edna with CryptDB [**?**], 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.

6.1.4 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.

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.

6.1.5 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 **??** shows that common operations have comparable latencies with and without Edna. Edna adds 9ms to account creation; and disguising and revealing 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 **??** 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 [**?**]. Content is distributed among users in a Zipf-like distribution according to statistics from the actual Lobsters deployment [**?**], 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 **??**. 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.

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. Edna's database operations are fast; in our prototype, they 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 disguise 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.

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 **??** 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.

6.1.6 Edna Overheads

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

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 disguise 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 removed principals, but stores encrypted diff records with their information, which uses another

2.2 MB.

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 [**?**].

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 **??** 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 transactions, high load). Under both low and high load, the throughput of normal operations is only slightly affected by concurrent disguising, dropping by at most 5.8% without transactions (and $< 17\%$ with transactions).

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 [**?**].

6.1.7 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 **??**) on the same WebSubmit operations (Figure **??**). 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 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., and anonymizing accounts. (lyt: note, one particular query during account anonymization for Qapla improved by an order of magnitude with index optimizations that we added when rerunning experiments, leading to comparable numbers to Edna for this operation.) 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.

6.1.8 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 [**?**]), 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 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.

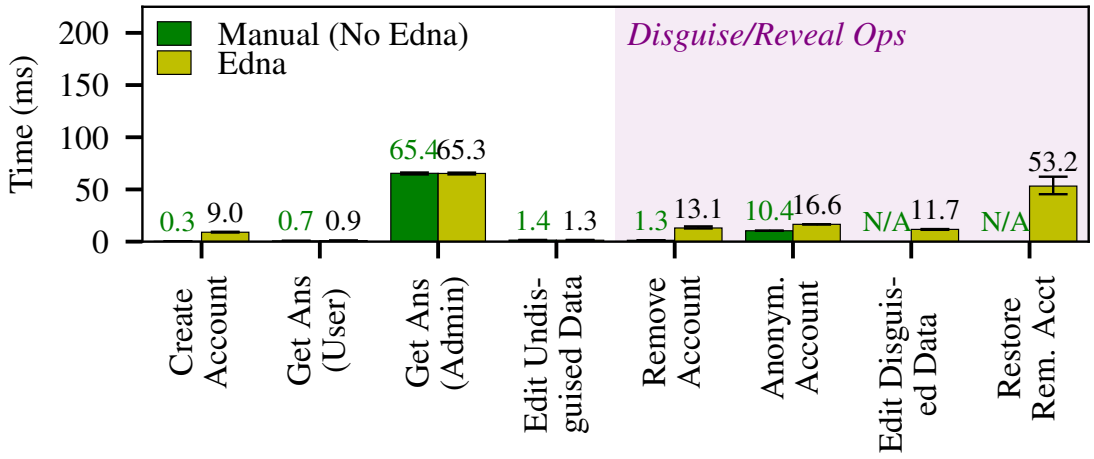
Performance. We measure the latency of WebSubmit operations like before, and compare a manual baseline, Edna, and Edna+CryptDB. Edna+CryptDB is necessarily more expensive than Edna, and a good result for Edna+CryptDB would therefore show moderate overheads over Edna, and acceptable absolute latencies.

Figure **??** 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.

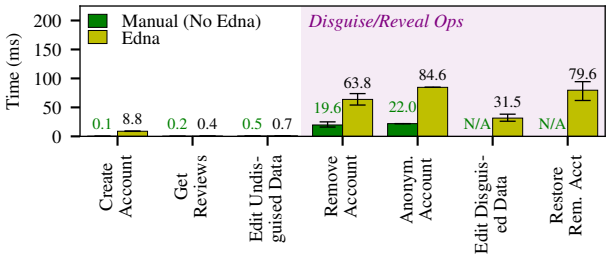
These overheads result from the cryptographic operations and additional indirection in Edna+CryptDB. Edna+CryptDB relies on a MySQL proxy, which adds latency: a no-op version of our proxy makes operations 1.03-1.5× slower. Cryptographic operations themselves are cheap (< 0.2ms), but every object inserted, 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).

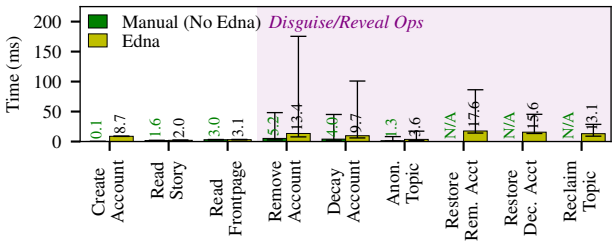
Like CryptDB, Edna+CryptDB increases the database size (4–5× for our WebSubmit prototype). Edna+CryptDB also stores an encrypted object key and its metadata (1KB per key) for each user with access to that object.



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

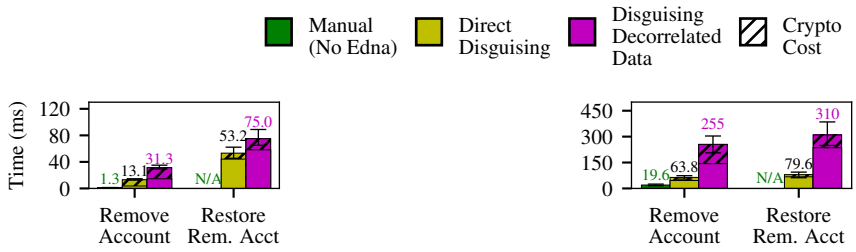


(b) HotCRP (80 reviewers, 3k total users, 200–300 records/reviewer).



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

Figure 6-1: 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.



(a) WebSubmit (b) HotCRP

Figure 6-2: 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.

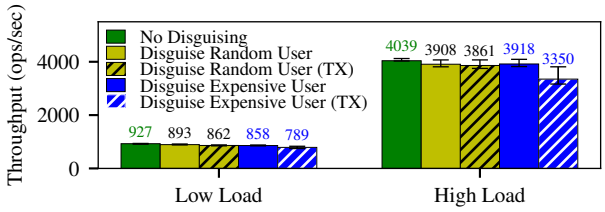


Figure 6-3: Continuous disguising/revealing operations in Lobsters have a <7% impact on application request throughput when disguising a random user; an extreme case of a heavy-hitter user with lots of data repeatedly disguising and revealing causes a 3–17% drop in throughput.

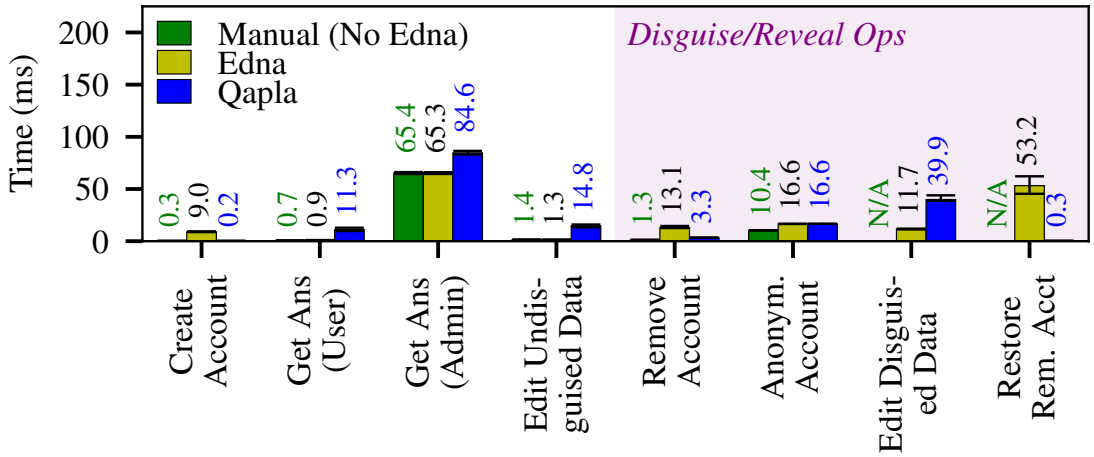


Figure 6-4: Edna achieves competitive performance with a manual baseline and outperforms Qapla on nearly all common WebSubmit operations on nearly all operations in WebSubmit (2k users, 80 answers/user). Bars show medians, error bars are 5^{th}/95^{th} percentile latencies.

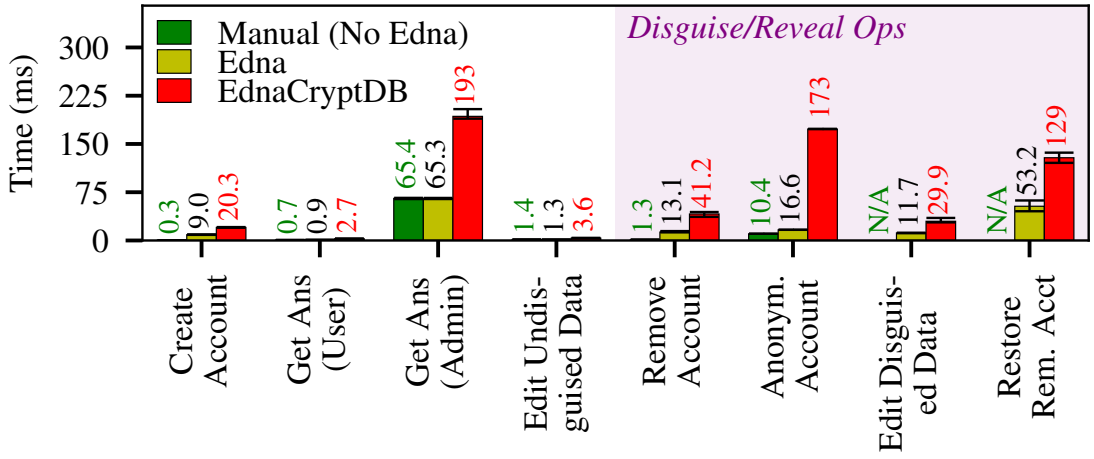


Figure 6-5: Latencies of WebSubmit (2k users, 80 answers/user) operations when implemented with Edna+CryptDB (adding encrypted database support). Bars show median latency; error bars are 5^{th}/95^{th} percentile latencies.

Chapter 7

Conclusion

Conclusion

7.1 Discussion

7.2 Future work