Edna: Disguising and Revealing User Data in Web Applications

Lillian Tsai Hannah Gross [†] M. Frans Kaashoek Eddie Kohler [‡] Malte Schwarzkopf [†]
MIT CSAIL [†] *Brown University* [‡] *Harvard University*

Abstract

Edna is a system that 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.

Experiments with Edna that add disguising and revealing transformations to three real-world applications show that Edna enables new privacy features in existing applications with low developer effort, is simpler than alternative approaches, and adds limited overhead to applications.

CCS Concepts: • **Security and privacy** → *Data anonymization and sanitization*; **Management and querying of encrypted data**; *Information accountability and usage control*; *Usability in security and privacy*.

Keywords: Web Applications, Data Privacy, Anonymization, Data Encryption, GDPR, PII

ACM Reference Format:
Lillian Tsai, Hannah Gross, M. Frans Kaashoek, Eddie Kohler, Malte Schwarzkopf. 2023. Edna: Disguising and Revealing User Data in Web Applications. In *ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP '23), October 23–26, 2023, Koblenz, Germany*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3600006.3613146

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 [9, 22, 54]. While users now have the right to delete this data (via e.g., the GDPR [21] or CCPA [8]), users want and deserve more nuanced controls over their data that don't exist today.

Consider Twitter: after a change in management [15], 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 [4, 36] 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 [3, 40, 48]. 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 [17, 39] 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 [44, 56]. 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 [43] and Instagram "finstas" [60]), 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.

1.1 Why is this hard?

Applications lack such functionality today 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

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SOSP '23, October 23–26, 2023, Koblenz, Germany
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0229-7/23/10.
https://doi.org/10.1145/3600006.3613146

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.

1.2 Our Approach

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.

Our approach is to create a general system that helps developers specify and apply two kinds of transformations: *disguising transformations*, which render all or some of the user's original sensitive data inaccessible to the application; 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.

1.3 Challenges

We had to address three challenges to make this approach work. 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, and unlocks and restores data to the service only when a user provides their credentials to reveal.

1.4 Contributions

In summary, this paper makes four key contributions:

1. 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.
2. Techniques to implement these abstractions, including pseudoprincipals (§4.2), speaks-for and diff records (§4.3), and speaks-for chains (§4.6).
3. Case studies that integrate Edna with three real-world web applications and demonstrate Edna's ability to enable composable and reversible transformations.
4. An evaluation of Edna's effectiveness and performance, including how Edna contrasts with and complements related work (Qapla [37] and CryptDB [45]).

While Edna enables disguising and revealing transformations in a broad class of applications, Edna has some limitations. First, Edna assumes bug-free disguise specifications, and that applications use Edna correctly. Second, while Edna helps developers add user data controls to single applications, Edna does not tackle the problem of data sharing between services. Third, 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.

2 Related Work

Edna is the first system to address the problem of reversible and composable data transformations for selective data removal 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 Edna's.

Data deletion tools, such as DELF at Meta [14], help correctly delete data. DELF lets developers specify deletion policies via annotations on social graph edges and object types, and ensures correct cascading data deletion. Other systems support wholesale user data deletion by tracking data ownership by modifying the data layout [16, 52] or tracking information flow [32]. While Edna also supports GDPR account deletion, Edna's focus is on more nuanced use cases beyond simple deletion: 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 [37] 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 [13, 24, 28, 50, 62], authorized views [6], per-user views [35], or by blocking or rewriting database queries [37, 42, 64]. Policy-enforcing systems do not help users anonymize data or maintain application integrity constraints, which is Edna's explicit goal. Instead of denying access to data, Edna changes the database contents so sensitive data is no longer available in the database, and thus unavailable even to the service itself.

Encrypted storage systems such as CryptDB [45] and Mylar [46] protect against database server compromise, with some limitations [26]. 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 [45], or move application functionality client-side [46]. Encrypted databases have orthogonal goals to Edna's: 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 Edna does. Any user with legitimate access can view the data in an encrypted database, whereas Edna removes disguised data from the database.

Other related work. Devices using iOS [2], Android [2], or CleanOS [55] revoke data access via encryption, like Edna does. However, these systems operate in settings that store only a single user's data; Edna instead tackles the problem of transformations that operate over multiple users' data and shared data without breaking the application.

Vanish [23] provides users with self-destructing data and a proof of data deletion using decentralized infrastructure and cryptographic techniques (with limitations against Sybil attacks [61]). Unlike Edna, Vanish cannot restore deleted data and requires extensive application restructuring.

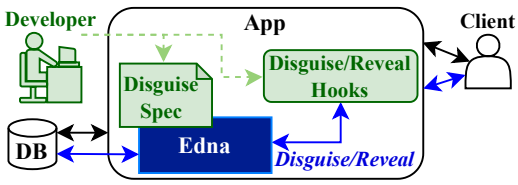


Figure 1. Developers write disguise specifications and add hooks to invoke Edna from the application (green); in normal operation, clients use these hooks in the application to disguise and reveal their data in the database (blue).

Sypse [18] 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.

Decentralized platforms such as Solid [49], BSTORE [12], Databox [38], and others [1, 10, 11, 33, 41] 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.

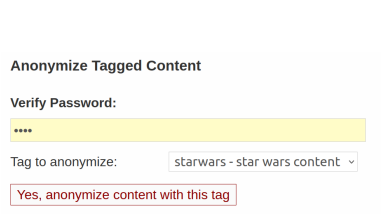
Some platforms can prove that server-side processing respects user-defined policies via cryptographic means [7] or systems security mechanisms [59]. This may restrict feasible application functionality (e.g., to additively homomorphic functions), or restrict combining data with different policies. Edna protects data only after disguising, but allows unrestricted application functionality before disguising.

3 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 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




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

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).

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 [19, 25, 47, 57].

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;[1] 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.

4 Design

We now describe how Edna's API and disguise specifications work via a disguising transformation for Lobsters [34].

4.1 Example: Lobsters Topic Anonymization

Lobsters [34] 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 (§4.2) and provide it to Edna. They also add frontend code and UI elements that allow authenticated users to trigger the disguising transformation (Figure 2). When Bea wants to anonymize their contributions on content tagged "Star Wars", Lobsters invokes Edna with a disguise specification that instructs Edna to decorrelate comments and remove votes on "Star Wars" stories (§4.3).

[1] If the application restores a backup, Edna continues operating as if only the transformations up to the time of the snapshot had been applied.

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

4.2 Disguise Specifications

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

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

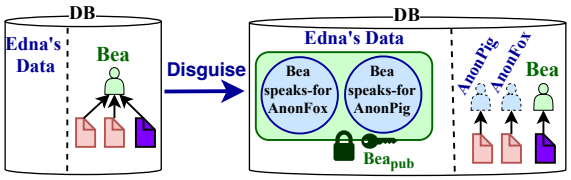


Figure 4. When Edna applies topic-based anonymization to Bea's comments on stories tagged "Star Wars" (red), these comments are decorrelated to pseudoprincipals ("AnonPig", "AnonFox") and Edna stores encrypted speaks-for records mapping Bea to their pseudoprincipals.

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.

4.3 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. Edna then performs the specified database changes by first applying all removals, and then decorrelations and modifications in specification order, potentially generating and storing pseudoprincipals.

Edna next generates *diff records* that contain the original data, the changes the disguise made to the original data (e.g., the value of any modified columns), and the disguise ID. For each new pseudoprincipal, Edna generates a public–private keypair and a *speaks-for record* that contains 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 (§4.6). Edna then encrypts diff and speaks-for records—collectively called *disguise 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), 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 speaks-for records that map Bea to the created pseudoprincipals, diff records containing Bea's votes on "Star Wars" stories, and diff records that document Bea's original ownership of "Star Wars"-tagged comments; (*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 disguise 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 disguise 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 disguise records, so an attacker cannot link principals to their records. At reveal time, Edna can efficiently find 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.

4.4 Revealing

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

Edna's reveal procedure (Figure 5) 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 first restores disguised data that was removed. Edna then reveals any modifications, and finally performs recorrelations using decrypted speaks-for records. 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 disguise records.

```
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 5. 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 (§4.6).

Edna's reveal semantics rely on consistency checks to handle database changes (e.g., application updates to undisguised data). Edna reveals data only if 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; and 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. 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. §8 describes ways to increase the scope of Edna's checks.

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 disguise records, and filter those records for those with the disguise ID. Edna then restores deleted votes and Bea's ownership of decorrelated comments.

4.5 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. 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: after Bea disguises the message, the message is owned by Chris and a pseudoprincipal; if Chris then disguises the message, Edna removes it. Either owner can reveal the message, which restores the message to the database and recorrelates the revealing user. Regardless of the reveal order, if all owners reveal the message, Edna returns the message to its original state.

4.6 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 disguise 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 disguise records, find all pseudoprincipals corresponding 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.

Finally, Edna must 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, 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 5), and if reveal operations happen out of order, Edna removes an intermediate link in the speaks-for chain.

4.7 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, and updates the database with the modification.

4.8 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.

5 Implementation

We implemented our Edna prototype in 7.9k lines of Rust.

API. An application can use the prototype if: (*i*) it uses a MySQL database; (*ii*) rows to disguise have direct foreign key relationships to a users table, where each user corresponds to a row of that table; (*iii*) all rows to disguise are owned by one or more principals; and (*iv*) 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 the prototype.

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 [5].

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 [53] 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) [29]. 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 [63], 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.

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.

6 Case Studies

This section evaluates Edna by using it 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 [34], WebSubmit [51], and HotCRP [31].

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 throwaway 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 §4.1, 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.

6.2 WebSubmit

We integrated Edna as a Rust library with WebSubmit [51]. 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 [58] 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 [31]. HotCRP currently retains past conference data forever and requires manual requests for account removal [30]. 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.

7 Evaluation

Our evaluation seeks to answer five 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 [37]), 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)

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 [37] using Qapla's query

rewriting and access control policies; and (*iii*) an integration of Edna with CryptDB [45], 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.

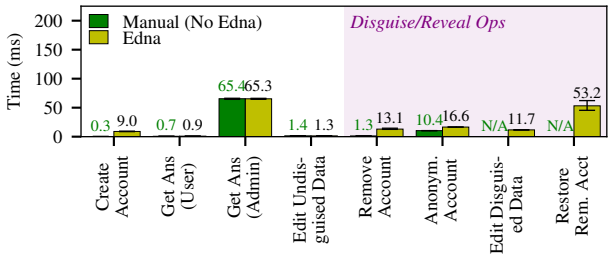
A developer using Edna also writes disguise specifications and invokes Edna. Lobsters' disguise specifications are written in 518 LoC, WebSubmit's in 75 LoC, and HotCRP's in 357 LoC (all in JSON). The specification size is proportional to schema size and what data each application disguises.

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.

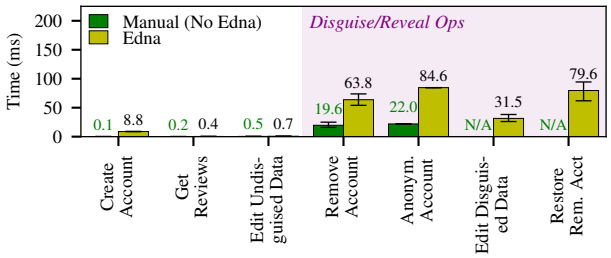
7.2 Performance of Edna Operations

We now evaluate Edna's performance using WebSubmit, HotCRP, and Lobsters (§6). 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)

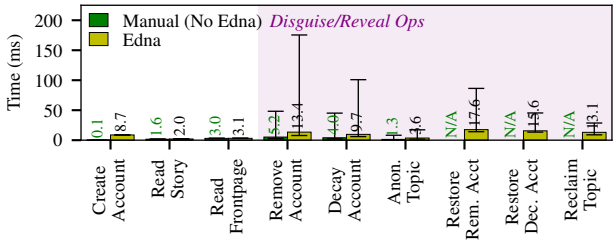
WebSubmit. We run WebSubmit with a database of 2k users, 20 lectures with four questions each, and an answer for each question for each user (160k total answers). We measure end-to-end latency to perform common application operations (which each issue multiple SQL queries), as well as disguising and revealing operations when possible (revealing operations are impossible in the baseline). Figure 6a shows that common operations have comparable latencies with and without Edna. Edna adds 9ms to account creation;



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



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



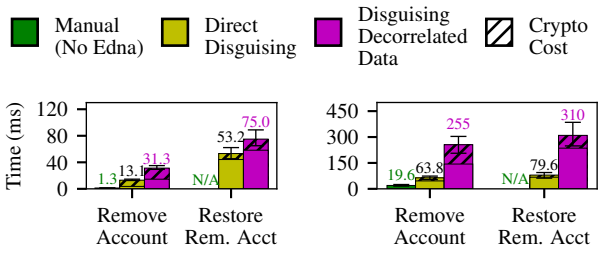
(c) Lobsters (16k users, Zipf-distributed data/user).

Figure 6. 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.

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 6b 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



(a) WebSubmit **(b)** HotCRP

Figure 7. 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.

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 [34]. Content is distributed among users in a Zipf-like distribution according to statistics from the actual Lobsters deployment [27], 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 6c. 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 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.

7.2.2 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 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 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.

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 [27].

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.

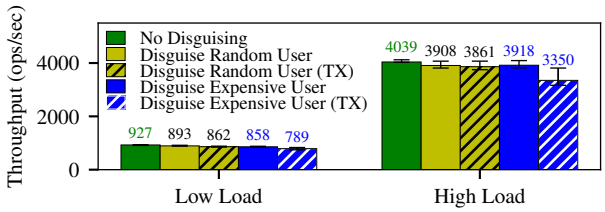


Figure 8. Continuous disguising/revealing operations in Lobsters have a <7% impact on application request throughput when disguising a random user; an extreme case of a heavy-hitter user with lots of data repeatedly disguising and revealing causes a 3–17% drop in throughput.

Figure 8 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).

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 [14].

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

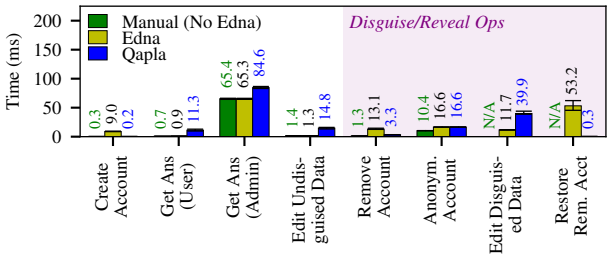


Figure 9. Edna achieves competitive performance with a manual baseline and outperforms Qapla on nearly all common WebSubmit operations (2k users, 80 answers/user). Bars show medians, error bars are 5^{th}/95^{th} percentile latencies.

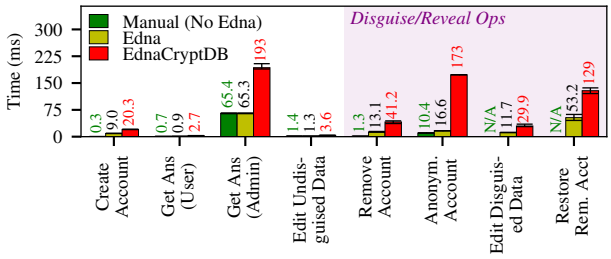


Figure 10. Latencies of WebSubmit (2k users, 80 answers/user) operations when implemented with Edna+CryptDB (adding encrypted database support). Bars show median latency; error bars are 5^{th}/95^{th} percentile latencies.

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 9) on the same WebSubmit operations (Figure 6a). 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. 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 [26]), 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 10 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.

8 Discussion and Future Work

Edna is a first step towards a world in which web services routinely manage, store, and reveal disguised user data. In this setting, new questions and directions for research arise.

Retention of Disguised Data. When a user reveals their data, Edna removes it from the disguise table. However, Edna currently retains disguised data until a user reveals it, which could be forever if users choose to never reveal data. 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.

Reveal Semantics. Edna today provides basic correctness guarantees when revealing data, but further work might make Edna's current reveal semantics more precise.

Consider an example: a user's disguise modifies posts to scrub their username from it, and a moderator later edits posts to remove swear words. As the disguise modifies the post, Edna today does not restore the original post content upon reveal, since the application has modified the post. However, Edna could restore the post if it knew how to subsequently remove the swear words again. Likewise, if Edna removed the post instead of scrubbing its content, the moderator would never see it (and could not edit it), but Edna would reveal the post with swear words still present. In the first scenario, Edna knows that an update was applied, and refuses to reveal the modified post; in the second, Edna does *not* know that moderation happened and reveals the removed post. Neither might be what the application desires.

Edna could handle this situation by tracking operations applied in a replay log. The application would invoke Edna when it performs operations that need to hold over revealed data—e.g., moderations or schema changes—to log these updates (as e.g., SQL queries) in Edna's replay log. When revealing data, Edna would apply every relevant entry in the replay log to the data about to be restored into the database. This approach faces some limitations, such as assuming deterministic changes and requiring additional application changes, and would need to ensure that the replay log can be stored and applied efficiently.

Pseudoprincipal references. Edna currently supports a global specification for checking and fixing references to pseudoprincipals. 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).

9 Conclusion

Edna enables developers to provide data disguising and revealing transformations that give users control over their data in web applications. 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 used Edna to add seven disguising transformations to three web applications, and 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.

Edna is open-source at https://github.com/tslilyai/edna.

Acknowledgments

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 [20] for providing resources for this paper's artifact evaluation.

References

[1] 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)*. Denver, Colorado, USA, June 2016, pages 181–194.

[2] Apple. *Apple Platform Security*. Apr. 2023. URL: https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf (visited on 05/2022).

[3] Daniel Arkin. *Celebrities are starting to leave Twitter. Here's a running list.* Oct. 31, 2022. URL: https://www.nbcnews.com/pop-culture/celebrity/twitter-celebrities-leaving-elon-musk-rcna54831 (visited on 04/13/2023).

[4] Daniel Augus. *Thinking of quitting Twitter? Here's the right way to do it.* Nov. 23, 2022. URL: https://economictimes.indiatimes.com/news/how-to/thinking-of-quitting-twitter-heres-the-right-way-to-do-it/articleshow/95696874.cms (visited on 07/01/2023).

[5] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. "Key-Privacy in Public-Key Encryption". In: *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Edited by Colin Boyd. Nov. 2001, pages 566–582.

[6] Kristy Browder and Mary Ann Davidson. "The Virtual Private Database in Oracle9iR2". In: *Oracle Technical White Paper, Oracle Corporation* 500.280 (2002).

[7] 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)*. July 2021, pages 387–404.

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

[9] Michelle Caruthers. *World Password Day: How to Improve Your Passwords*. Apr. 2023. URL: https://blog.dashlane.com/world-password-day/ (visited on 05/11/2018).

[10] 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.

[11] 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.

[12] 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)*. Boston, Massachusetts, USA, 2010.

[13] Adam Chlipala. "Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications". In: *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Jan. 2010, pages 105–118.

[14] 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)*. Aug. 2020.

[15] Kate Conger and Lauren Hirsch. *Elon Musk Completes $44 Billion Deal to Own Twitter*. Oct. 27, 2022. URL: https://www.nytimes.com/2022/10/27/technology/elon-musk-twitter-deal-complete.html (visited on 04/13/2023).

[16] 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.

[17] Vincent De Beer. *Heartbreak and Hacking: Dating Apps in the Pandemic*. Apr. 2021. URL: https://securityboulevard.com/2021/04/heartbreak-and-hacking-dating-apps-in-the-pandemic/ (visited on 08/23/2023).

[18] 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, Jan. 2021.

[19] 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-including-gdpr/ (visited on 04/08/2023).

[20] 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, Snigdhaswin Kar, and Prabodh Mishra. "The Design and Operation of CloudLab". In: *Proceedings of the USENIX Annual Technical Conference (ATC)*. July 2019, pages 1–14.

[21] "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)". In: *Official Journal of the European Union* L119 (May 2016), pages 1–88.

[22] 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)*. Banff, Alberta, Canada, 2007, pages 657–666.

[23] 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)*. Montreal, Canada, 2009, pages 299–316.

[24] 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)*. Hollywood, California, USA, Oct. 2012, pages 47–60.

[25] GlobalData Thematic Research. *Crypto-shredding the best solution for cloud system data erasure*. Jan. 2022. URL: https://www.verdict.co.uk/crypto-shredding-gdpr-cloud-systems/ (visited on 04/08/2023).

[26] 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)*. Whistler, British Columbia, Canada, 2017, pages 162–168.

[27] Peter Bhat Harkins. *Lobste.rs access pattern statistics for research purposes*. Mar. 2018. URL: https://lobste.rs/s/cqnzl5/lobste_rs_access_pattern_statistics_for#c_hj0r1b (visited on 03/12/2018).

[28] Katia Hayati and Martín Abadi. "Language-Based Enforcement of Privacy Policies". In: *Proceedings of the International Workshop on Privacy Enhancing Technologies*. Springer, 2004, pages 302–313.

[29] Burt Kaliski. *PKCS# 5: Password-based cryptography specification version 2.0*. Technical report. 2000.

[30] Eddie Kohler. *HotCRP.com privacy policy*. Aug. 2020. URL: https://hotcrp.com/privacy (visited on 12/07/2020).

[31] Eddie Kohler. *HotCRP.com*. URL: https://hotcrp.com (visited on 02/03/2021).

[32] 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*. Springer, 2019, pages 24–38.

[33] 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, Nov. 2007.

[34] *Lobsters*. URL: https://lobste.rs (visited on 02/03/2021).

[35] 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)*. 2019, pages 88–95.

[36] Cecily Mauran. *So you want to leave Twitter: A wellness plan*. Oct. 28, 2022. URL: https://mashable.com/article/how-to-leave-twitter-guide-elon-musk (visited on 07/01/2023).

[37] 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)*. Vancouver, British Columbia, Aug. 2017, pages 1463–1479.

[38] 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*. 2016, pages 49–54.

[39] Mozilla Foundation. *Tinder (*Privacy Not Included)*. Mar. 2021. URL: https://foundation.mozilla.org/en/privacynotincluded/tinder/ (visited on 08/23/2023).

[40] Ken Olin. *Hey all - I'm out of here. No judgement...* Oct. 28, 2022. URL: https://twitter.com/kenolin1/status/1586031904007954433 (visited on 04/13/2023).

[41] Shoumik Palkar and Matei Zaharia. "DIY Hosting for Online Privacy". In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets)*. 2017, pages 1–7.

[42] Primal Pappachan, Roberto Yus, Sharad Mehrotra, and Johann-Christoph Freytag. "Sieve: A Middleware Approach to Scalable Access Control for Database Management Systems". In: *arXiv preprint arXiv:2004.07498* (2020).

[43] Pranay Parab. *How to Make a Burner Account on Reddit, Even Though They Don't Want You to Anymore*. Jan. 2022. URL: https://lifehacker.com/how-to-make-a-burner-account-on-reddit-even-though-the-1848336857 (visited on 12/08/2022).

[44] Nicole Perlroth, Amie Tsang, and Addam Satariano. *Marriott Hacking Exposes Data of Up to 500 Million Guests*. Nov. 2018. URL: https://www.nytimes.com/2018/11/30/business/marriott-data-breach.html (visited on 12/19/2020).

[45] 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)*. Cascais, Portugal, 2011, pages 85–100.

[46] 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)*. Seattle, Washington, USA, 2014, pages 157–172.

[47] 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)*. Bellevue, Washington, USA, Aug. 2012, pages 333–348.

[48] Shonda Rhimes. *Not hanging around for whatever Elon has planned. Bye.* Oct. 29, 2022. URL: https://twitter.com/shondarhimes/status/1586399694896390147 (visited on 04/13/2023).

[49] 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.

[50] David Schultz and Barbara Liskov. "IFDB: Decentralized Information Flow Control for Databases". In: *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*. Apr. 2013, pages 43–56.

[51] Malte Schwarzkopf. *websubmit-rs: a simple class submission system*. URL: https://github.com/ms705/websubmit-rs (visited on 06/03/2020).

[52] Malte Schwarzkopf, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. "Position: GDPR Compliance by Construction". In: *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Cham: Springer International Publishing, 2019, pages 39–53.

[53] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* 22.11 (Nov. 1979), pages 612–613.

[54] 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)*. Menlo Park, California, USA, 2014, pages 243–255.

[55] 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)*. Hollywood, California, USA, Oct. 2012, pages 77–91.

[56] The Office of the Privacy Commissioner. *New Zealand's biggest data breach shows retention is the sleeping giant of data security*. Apr. 2023. URL: https://www.privacy.org.nz/publications/statements-media-releases/new-zealands-biggest-data-breach-shows-retention-is-the-sleeping-giant-of-data-security/ (visited on 12/01/2022).

[57] Patrick Townsend. *GDPR, Right of Erasure (Right to be Forgotten), and Encryption Key Management*. Jan. 2028. URL: https://info.townsendsecurity.com/gdpr-right-erasure-encryption-key-management (visited on 04/08/2023).

[58] U.S. Department of Education. *FERPA*. Aug. 1974. URL: https://studentprivacy.ed.gov/ferpa.

[59] 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)*. Boston, Massachusetts, USA, Feb. 2019, pages 615–630.

[60] Caity Weaver and Danya Issawi. *'Finsta,' Explained*. Sept. 2021. URL: https://www.nytimes.com/2021/09/30/style/finsta-instagram-accounts-senate.html (visited on 12/08/2022).

[61] 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, Feb. 2010.

[62] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. "A Language for Automatically Enforcing Privacy Policies". In: *ACM SIGPLAN Notices* 47.1 (2012), pages 85–96.

[63] 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*. San Francisco, California, USA: Springer, 2005, pages 245–261.

[64] 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)*. Carlsbad, California, USA, July 2022, pages 701–718.