Edna: Helping Web Developers Give Users Control Over Their Data

Paper #243

Abstract. Users today give web applications their data but have little control over how these applications store and protect this data. Giving users control is challenging for developers, as application functionality sometimes requires the very same data that users want to protect.

Edna is a library that lets web application developers give users flexible and correct control over their data's exposure without breaking the application. Edna minimizes application changes and developer effort, and enables applications to securely hide user data and users to later restore it. Edna introduces abstractions for *sealing*, which selectively renders user data inaccessible via encryption, and *revealing*, which enables the owning user to restore their data to the application.

We used Edna to add user data control to three real-world applications. We show that Edna provides flexible user data control, is simpler than alternative approaches, and achieves practical performance.

1 Introduction

Users today have little to no control over the data they give to web applications. They must either avoid using an application, or leave their data susceptible to leaks, e.g., via SQL injections or compromise of other users' accounts. Incentivized by growing public demand and laws such as the GDPR and CCPA [7, 14], developers increasingly need to implement mechanisms to give users control over their data's exposure.

Central to this conversation are two needs: users want to protect their data, and applications require data to be available in order to function. Problems arise when application functionality requires the same data users want to protect: for example, deleting a post may orphan other users' comments on it, crashing the application when it expects comments to have an associated post. Handling this correctly is difficult for developers—naively removing user data can break the application or reduce its functionality. Removal is also usually permanent, hurting both users and applications: once a user deletes their data, there is no coming back.

Large companies have built systems to help developers get data deletion right [12], but the problem goes beyond data deletion. Users may want to take a break from using an application, but rest easy that their data is protected while they're inactive; or users may want more fine-grained control—i.e., have their data exist in a state where it is both visible and protected. For example, a user might want to disassociate content with a specific hashtag from a social media account. Because applications today lack support for this, users resort to ad-hoc alternatives like throwaway accounts and "finstas" [30, 41] to post content that is visible but disconnected from the "real" account. This burdens users with managing multiple accounts and requires planning ahead, since moving posts into a throwaway account or reclaiming them is impossible.

Edna is the first system that lets web application developers provide users with means to manage their data in the gray space where users' need for data protection and application functionality requirements overlap. Edna integrates into the web application as a library and provides abstractions that let developers specify not just wholesale removal of data, but also flexible redaction or decorrelation of data. Edna then changes the database contents in well-defined ways that avoid breaking the application. Developers and users both benefit from Edna: applications can expose and advertise new privacy-protection features and reduce their exposure in the event of a data breach, while users gain more control over their sealed data, including the convenience of returning to the application as if their data had been there all along.

Edna introduces *sealing*, which removes or redacts some or all of a user's data, and *revealing*, which restores the sealed data at a user's request. Sealed data remains on the server, but is encrypted and inaccessible to the web application. Sealing changes the database contents and replaces the data to seal with placeholder values where necessary (e.g., comments require an associated post).

We designed Edna's seal and reveal abstractions to flexibly meet applications' diverse needs while still protecting sensitive user data. Edna provides developers with three well-defined primitives for anonymizing data: remove, modify, and decorrelate. Developers compose these to create *seal specifications*, which describe the data to seal and which primi-

tives to apply. When invoked with a seal specification, Edna changes the database contents as specified: "remove" drops rows, "modify" changes the contents of specific cells, and "decorrelate" introduces *pseudoprincipals*, anonymous placeholder users. These pseudoprincipals help maintain referential integrity, but also can act as built-in "throwaway accounts." Rather than users having to create and maintain throwaways themselves, Edna's pseudoprincipals enable the user to later reassociate with their throwaway's data via revealing, or create throwaways to disown posts after-the-fact.

Edna is complementary to existing data protection mechanisms. For example, we combine Edna with an encrypted database to achieve stronger guarantees. Edna-CryptDB simultaneously protects against database server compromises even for unsealed data and adds Edna's protections for sealed data to encrypted databases, which have no built-in support for removing sensitive data without breaking the application.

To investigate the need for Edna as a new system, we tried to realize Edna's functionality atop Qapla [25], a framework that rewrites SQL queries to conform to access control policies. We found that Qapla requires invasive application changes, its abstractions are awkward for sealing and revealing, and Qapla's query rewriting slows down common queries.

We implemented a prototype of Edna, and evaluated it with three real-world web applications: Lobsters, a discussion forum; HotCRP, a conference review system; and WebSubmit, a homework submission application. Integrating Edna into these applications required minor code changes. For example, supporting three types of sealing in the 160k-LoC Lobsters application required adding 697 LoC. The modified applications retained their expected semantics and functionality. Edna's sealing and revealing performance depends on the amount of data affected, but they generally complete in seconds and have a small effect on the performance of concurrent operations.

This paper makes the following contributions:

1. A new paradigm for developers to provide flexible user control over data in web applications, with cryptographic protection and while maintaining application functionality and invariants.
2. Abstractions for sealing and revealing, and a small set of data-anonymizing primitives (remove, modify, decorrelate) that cover a wide range of application needs and compose cleanly.
3. The design and implementation of Edna, our prototype library that implements user data control via sealing and revealing, as well as Edna-CryptDB, which additionally encrypts unsealed data.
4. Case studies of integrating Edna with three real-world web applications, an evaluation of Edna's effectiveness, performance, and security, and a comparison to Qapla.

Edna has some limitations. Its usability goals, and specifically the fact that placeholder data continues to exist in the database, prevent Edna from protecting against inference attacks (e.g., statistical correlation attacks). Edna also assumes bug-free seal specifications, and that applications use Edna correctly. Finally, Edna provides limited protection for sealed data against attackers who compromise the database server, as auxiliary database information (e.g., logs) may contain old plaintext data; Edna-CryptDB strengthens these protections.

System	**User u's data is protected against...**		
	SQL injection	*Compromised user $\neq u$*	*Server compromise*
Qapla [25]	✓		
CryptDB [31]	✓		✓
Edna	✓	✓	
Edna-CryptDB	✓	✓	✓

Figure 1: Threats protected against by different classes of systems. Edna also keeps the application functional if it relies on data that users want to protect.

2 Background and Related Work

Edna gives developers tools to remove and redact user data from a web application without breaking the application. Existing systems for data protection complement Edna, as they aim instead to support data deletion, prevent unauthorized data access, or protect against database server compromise.

Laws and regulations such as the EU's General Data Protection Regulation (GDPR) [14], the California Consumer Privacy Act (CCPA) [7], and others [37, 38] give users the right to request deletion of their data from web applications. **Data deletion tools** developed at large organizations, such as DELF at Meta [12], help to correctly delete data in response to such requests. DELF helps developers specify correct deletion policies via annotations on social graph edges and object types, and ensures correct cascading data deletion.

Like data deletion systems, Edna supports GDPR account deletion. However, Edna also supports use cases beyond simple deletion. Edna can seal data in a GDPR-compliant manner and allow the user to return, seal old data for inactive users, or seal some but not all data, allowing the user to continue using the application.

Policy enforcement systems such as Qapla [25] aim to prevent unauthorized access to data and protect against leakage via compromised, unauthorized accounts or SQL injections. They enforce developer-specified visibility and access control policies via information flow control [11, 15, 18, 34, 42], authorized views [5] or per-user views [24], or blocking or rewriting database queries [25, 29, 44].

Encrypted storage systems such as CryptDB [31] and Mylar [32] aim to protect against database server compromise, with some limitations [16]. These systems encrypt data in the database, and ensure that only users with legitimate reasons to view the data (i.e., the owner or an admin) can decrypt the data. Applications handle keys, and send queries either through trusted proxies that decrypt data [31], or move application functionality client-side [32].

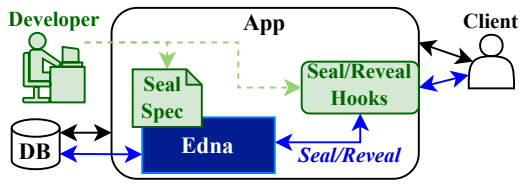


Figure 2: Developers write seal specifications and add hooks to invoke Edna from the application (green); in normal operation, clients use these hooks in the application to seal and reveal their data in the database (blue).

Policy-enforcing and encrypted storage systems do not help users anonymize data or maintain application functionality in the face of anonymized data, which is Edna's explicit goal. Unlike these systems, Edna changes the database contents so sensitive data is no longer available in the database. Edna encrypts users' sealed data, and ensures that the key material is unavailable to the application and any privileged account. Edna can leverage policy-enforcement or encrypted storage to strengthen its protections for unsealed data; we explore this with Edna-CryptDB. Figure 1 shows the threats that policy-enforcement systems (e.g., Qapla [25]), encrypted databases (e.g., CryptDB [31]), and Edna defend against.

Other Related Work. Some systems modify the data layout [35] or use fine-grained information flow tracking [21] to track the ownership and provenance of database rows. This tracks how information propagates and provides information to support wholesale user data deletion. Edna requires more developer input, but supports nuanced transformations like decorrelation or partial anonymization of shared data.

Sypse [13] pseudonymizes user data and partitions personally-identifying information (PII) from other data. Instead of partitioning data, Edna actually modifies the application database and stores sealed data encryptedly.

Decentralized platforms such as Solid [33], BSTORE [10], Databox [26], and others [2, 8, 9, 22, 28] put data fully and directly under user control, since users store their own data. But this burdens users with maintaining infrastructure, and decentralized platforms 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, storing all data, including sealed data, on the application's servers.

Some platforms can prove that server-side processing respects user-defined data policies via cryptographic means [6] or systems security mechanisms [40]. This may restrict feasible application functionality (e.g., to additively homomorphic functions), or restrict combining data with different policies. Edna only protects user data after sealing, but in exchange allows unrestricted application functionality before sealing.

3 Edna Overview

Edna supports user data control in web applications using application-specific *sealing transformations*. To integrate an application with Edna, the developer writes seal specifications and adds hooks to seal or reveal data using Edna's API (Figure 2). Applications (1) register users (human principals) with Edna, (2) invoke Edna to apply transformations that seal user data, and (3) invoke Edna to reveal that data when requested.

(1) Applications register 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 remembers the private key for use in future reveal operations.

(2) When the application wants to seal some data, it invokes Edna with the corresponding developer-provided seal specification and any necessary parameters (such as a user ID). Seal 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 connection between the user and any pseudoprincipals it created, encrypts that data with the user's public key, and stores the resulting ciphertext—the *sealed data*—such that it cannot be linked to the user without the user's private key.

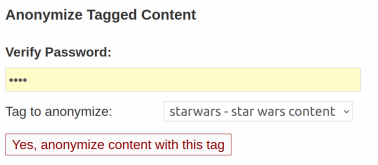
(3) When a user wishes to reveal their sealed data, they pass credentials to the application, which calls into Edna to reveal the data. Credentials are application-specific: users may either provide their private key or other credentials sufficient for Edna to re-derive the private key. Edna gets the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

Edna provides the developer with sensible default sealing and revealing semantics (e.g., revealing makes sure not to overwrite changes made since sealing), which the developer can customize via a low-level API provided by Edna.

Threat Model. Edna assumes well-intentioned developers who write seal specifications that capture the desired application semantics. Edna seals data according to these specifications, but because some data must be retained in the database to keep application semantics intact, Edna cannot protect against inference attacks. Nevertheless, Edna protects a user's *sealed* data against:

1. arbitrary application database accesses (e.g., SQL injection attacks);
2. public exposure or other users' ability to view the data through the application (e.g., compromised accounts, including privileged ones); and
3. remote code execution with the application's privileges (e.g., PHP's `eval()`).

Edna guarantees that a user's sealed data is protected unless an attacker obtains their credentials. While Edna hides the contents of sealed data and relationships between sealed data and users, it does not hide the existence of sealed data. (An attacker can see whether a user has sealed some data, but cannot see which sealed data corresponds to this user.) Some transformations, especially decorrelation, cannot protect against




```
def tag_anon
  # instantiate the seal spec with the provided tag to anonymize
  seal_spec = edna.instantiate_spec("tag_anon.json", params[:tag])
  # apply the sealing transformation
  seal_id = edna.apply_seal(@user.id, params[:password], seal_spec)
  # email the seal ID to the user to allow revealing
  SendSealEmail(@user, seal_id)
end
```

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

statistical inference attacks over information still visible in the database. Edna also cannot handle copies of data embedded in other data (e.g., quoted text) or snapshots saved when the data was still in the database. We make standard assumptions about the security of cryptographic primitives: attackers cannot break encryption, and keys stored with clients are safe.

4 Design

We now describe how Edna's API and seal specifications work in the context of a sealing transformation for Lobsters [23].

4.1 Example: Lobsters Category Anonymization

Lobsters [23] is a link-sharing and discussion platform with 15.4k users. Its schema consists of stories, tags on stories, comments, votes, private messages, user accounts, and other user-associated metadata. Clients create accounts, submit URLs as stories, and interact with other users and their posted stories via comment threads and votes.

Consider **category-based content anonymization**, which allows users to hide their interest in a topic category (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 form of user data control is currently not possible in Lobsters.

Edna realizes category-based content anonymization for Lobsters as a sealing transformation. In the first step, the Lobsters developers write a seal specification (§4.2) and provide it to Edna. They also add frontend code and UI elements that allow authenticated users to trigger the sealing transformation (Figure 3). When Bea wants to anonymize their contributions on content tagged "Star Wars," Lobsters invokes Edna with a seal specification that instructs Edna to decorrelate comments and remove votes on "Star Wars" stories (§4.3).

4.2 Seal Specifications

Seal specifications tell Edna what application data objects to seal and how to seal them. Objects are specified by database table name, principal, and predicate, where a predicate is a SQL `WHERE` clause. Edna by default seals all objects related to the given principal, as defined by a foreign-key relationship provided in the seal 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,

```
// 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 4: Lobsters category-based content anonymization seal specification (JSON pseudocode).

developers choose to *remove*, *modify*, or *decorrelate* them. The example specification in Figure 4 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 seal specification may tell Edna to create a unique pseudoprincipal for each decorrelated application object. In our example, however, all comments on the same story decorrelate to the same pseudoprincipal (`"group_by": "stories.id"`), thus keeping same-story comment threads intact. 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 (e.g., creating a "Star Wars" persona).

Applications can use Edna to generate pseudoprincipals, or can generate pseudoprincipals themselves and provide Edna with these pseudoprincipals' unique IDs.

4.3 Sealing

To apply a sealing transformation, Edna creates a unique *seal ID* and queries for the data to seal based on the seal specification predicates. Edna then performs the specified database changes by first applying all decorrelations and modifications

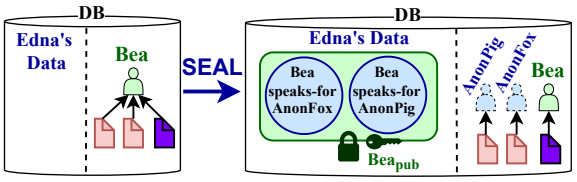


Figure 5: When Edna applies category-based content anonymization to Bea's comments on stories tagged "Star Wars" (red), these comments are decorrelated to pseudoprincipals ("AnonPig", "AnonFox") and Edna stores encrypted speaks-for records mapping Bea to their pseudoprincipals.

in specification order (potentially generating and storing pseudoprincipals), and then removals.

Edna next generates *diff records* that contain the original data and the seal ID. For each new pseudoprincipal, Edna generates *speaks-for records* that document the relationship to the original principal and the pseudoprincipal's private key. Edna encrypts and stores diff and speaks-for records—collectively called *seal records*—in the database. Finally, Edna returns the seal ID to the application. The seal ID and the principal's private key can be used to reveal the transformation later.

To perform Bea's category-based anonymization (Figure 5), 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, and diff records containing Bea's original votes on "Star Wars" stories; (*v*) encrypts the speaks-for and diff records with Bea's public key, stores them; and (*vi*) returns a unique seal ID to the application.

Edna adds a *seal table* and a *principal table* to the application database to securely store principals' sealed data. The seal table contains lists of per-principal seal 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 seal table indexes encrypted with the public key. Edna stores seal records for principal p by (1) encrypting the records with p's public key; (2) storing the ciphertext in the seal table under index `idx`; and (3) appending [`idx` $\|$ random-nonce] (encrypted with p's public key) to p's list of encrypted seal tables indexes in the principal table. This allows Edna to store records without needing the principal's private key, and to do so securely: the principal table adds a layer of indirection from user ID to encrypted seal records, so an attacker cannot link principals to their records. At reveal time, Edna can efficiently find sealed data for a given user by decrypting and using seal table indexes in the principal table.

A series of sealing transformations may completely remove a principal from the application database. When this happens,

```
Reveal(sealID, uid, privkey):
 encrypted_seal_table_idxs := principal_table[uid]
 decrypted_seal_table_idxs :=
    decrypt(encrypted_seal_table_idxs, privkey)
 for idx in decrypted_seal_table_idxs:
   records = decrypt(seal_table[idx], privkey)
   for rec in records:
     if rec.sealID==sealID:
       // apply rec to application database
       // remove rec from seal_table
     else if rec.type==SPEAKS_FOR:
       // recursively reveal for pseudoprincipal
       Reveal(sealID, rec.pp_uid, rec.pp_privkey)
```

Figure 6: Pseudocode for revealing a seal transformation (while application principal `uid` exists).

Edna moves the corresponding list of encrypted seal table indexes from the principal table to a *deleted principal table* indexed opaquely by, e.g., the public key. This removes the user ID from the database while allowing future reveal operations by the principal to find their seal table indexes.

4.4 Revealing

To apply a reveal transformation, Edna first locates and decrypts the corresponding seal records using a seal ID and the user's *reveal credentials*. Edna supports two forms of reveal credentials: (*i*) the principal's private key itself; or (*ii*) backup secrets, such as a strong application password or a recovery token, which can be used 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. To support forgetful users, Lobsters could use a backup token. Password or keypair changes require an application to re-register the user with Edna, which generates new backup credentials.

The reveal procedure (Figure 6) first looks up all seal records related to the provided reveal credentials via Edna's principal and seal tables. Edna applies records created for the `sealID` seal transformation to the database, thus restoring the relevant application objects to their pre-sealed state.

To preserve referential integrity, Edna first restores sealed 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 data associated, removing them from the principal table and the application's users table. After revealing a sealing transformation, the sealed data is no longer needed, so Edna clears the corresponding seal records.

Edna provides sensible default reveal semantics to handle applications whose databases are changing. Edna reveals data in its unsealed, original state if four consistency checks pass:

1. *removed* sealed data is still removed from the database (no rows conflict in uniquely identifying columns);
2. *modified* sealed data is in the same modified state;
3. *decorrelated* sealed data is still affiliated with the correct

pseudoprincipal in the database; and
4. the data's *foreign key references* exist in the database.

This ensures referential integrity and adherence to column uniqueness constraints.

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

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 seals shared data, Edna decorrelates the data from the sealing user, but preserves the data and its association with other owners. The data is removed once all users have sealed it and all ownership links are to pseudoprincipals. For instance, consider a Lobsters message between Bea and Chris: after Bea seals the message, the message is owned by Chris and a pseudoprincipal; if Chris then seals the message, it is removed. Either owner can reveal the message, which restores the message to the database and recorrelates the revealing user. If all owners reveal the message, the message is restored to its original state, regardless of the reveal order.

4.6 Composing Sealing Transformations

Edna naturally supports sealing composition, where a sealing transformation is applied to data that has already been sealed in some other way. For instance, a user could anonymize some posts, after which an administrator could choose to anonymize *all* posts. In this scenario, the administrator's sealing 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.

Another interesting case is when a transformation should apply to data that a user owned, but that has already been decorrelated. For instance, a Lobsters user might first anonymize (decorrelate) some of their comments and then request to delete all their comments. 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 seal operation. These credentials let Edna decrypt the user's previous seal records, find all pseudoprincipals corresponding to that user, and apply sealing transformations on behalf of those pseudoprincipals as well as the requesting user.

Having reveal credentials at sealing time also supports another desirable form of sealing composition: data that can be decorrelated multiple times, but should not be recorrelated until all seals are removed. For example, if Bea separately seals and decorrelates their comments on "bears" and "Star Wars" posts, then later reveals the "bears" posts, they might want Ewok-related comments (which are tagged both "Star Wars" and "bears") to remain sealed, even though they were initially sealed under the "bears" transformation. With reveal credentials, Edna finds pseudoprincipals with already-decorrelated comments and applies the sealing transformation to these pseudoprincipals. This again creates pseudoprincipals that can speak-for other pseudoprincipals.

Pseudoprincipal-to-pseudoprincipal speaks-for relationships requires special handling during reveal operations: Edna internally maintains the correct chain of speaks-for records even if reveal operations are applied out of order.

4.7 Authenticating As Pseudoprincipals

As described so far, if Bea wanted to modify a "Star Wars" comment they had previously anonymized, they would have to reveal the comment, edit it using their normal credentials, and then seal the comment again. 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; using these credentials, Edna validates that the user speaks-for a specific pseudoprincipal, and updates its seal records to reflect the modification.

4.8 Edna-CryptDB

To harden Edna's security guarantees, developers can deploy an Edna-enabled application atop an encrypted database such as CryptDB [31]. This protects unsealed database contents against attackers who compromise the database server itself (with some limitations [16]), in addition to Edna's existing protections for sealed data. We refer to the combination of Edna and CryptDB as Edna-CryptDB.

A developer using Edna-CryptDB deploys the application (and Edna) atop a proxy that encrypts and decrypts database rows. Edna-CryptDB operates on unsealed data in the same way as CryptDB, and operates in CryptDB's threat model 2, in which database server and proxy can be compromised.

Developer Experience. To use Edna-CryptDB, application developers specify which users can access what data (e.g., by specifying foreign keys that indicate ownership of objects). Queries from Edna and the application operate unchanged atop the proxy. However, to ensure proper access to user data, the application must handle user sessions and provide the proxy their private keys. Edna-CryptDB exposes an API that lets the application log in and log out users. This handles key storage and removal in the same way as CryptDB.

Sealing and Revealing. Because the database is encrypted, sealing operations in Edna-CryptDB can only operate on data that logged-in users can access. Thus, sealing a user's data must run while someone with legitimate access to the user's data (e.g., the user, an admin, or someone sharing the data) is logged in. Edna-CryptDB requires developers to ensure that their seal specifications touch only data accessible to the

user who invokes the sealing transformation. In addition, if a user's records during sealing or revealing include speaks-for records, Edna-CryptDB logs in the pseudoprincipals with the private keys encoded in the speaks-for records.

Edna-CryptDB performs data removal and data modification identically to Edna, aside from the initial login of users. Decorrelation, however, requires slightly more nuance: for each new pseudoprincipal p produced, Edna must ensure that a client speaking for p can access both p's account object (e.g., a `users` table row) and the objects correlated to p. To do so, Edna encrypts (with p's key) the keys associated with p's account and any correlated objects.

5 Implementation

We implemented our Edna prototype in 6.7k lines of Rust (1.6k LoC specifically added for Edna-CryptDB).

API. Edna offers a high-level API and a low-level API. The low-level API is independent of the database used and gives applications control over how sealed data is organized.

§4 described the high-level API. An application can use the high-level API if: (*i*) it uses a MySQL database; (*ii*) rows to seal have direct foreign key relationships to users table, where each user corresponds to a row of that table; (*iii*) all rows to seal are owned by one or more principals; and (*iv*) all rows can be uniquely identified (e.g., via primary key). Edna stores its data (seal and principal tables) in the database.

With the low-level API, applications can run custom checks during revealing and apply bespoke sealing and revealing semantics for shared data (e.g., removing data once *any* owner seals it). However, the low-level API requires applications to generate pseudoprincipals, create diff and speaks-for records, and make database changes during sealing and revealing.

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 [4], allowing an attacker to determine which seal records belong to the same principal, but this is not fundamental.

Reveal Credentials. Our prototype supports two forms of reveal credentials: (*i*) the principal's private key itself; or (*ii*) passwords and recovery tokens as backup secrets. If an application chooses to use the latter, they provide the principal's password to Edna upon user registration. Our prototype uses a variant of Shamir's Secret Sharing [36] 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) [1]; Edna stores the resulting $f(x)$ half of the share. Thus, Edna can derive a full share when given only the user's password. Edna returns one full share as a recovery token and stores the remaining share; the private key can thus be reconstructed from either the user password or 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 sealing and revealing transformations in transactions, providing snapshot 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 certain 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.

Edna-CryptDB. Our prototype only supports the CryptDB deterministic encryption scheme and encrypts objects using AES-CMC, which limits it to equality comparison predicates. Our implementation uses per-object keys, and since these differ across tables, it does not support `JOIN`s, a limitation shared with multi-principal CryptDB (threat model 2).

6 Case Studies

We wrote sealing transformations for three applications—Lobsters [23], WebSubmit [3], and HotCRP [19]—and fully integrated Edna with Lobsters and WebSubmit. All applications use Edna's high-level API and Edna's default semantics.

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 sealing transformations.

First, **GDPR-compliant account removal** (*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 users remove their accounts, 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 sealing transformation, Lobsters emails the user a URL that embeds the seal ID. The user can visit this URL and provide their credentials to restore their account. This transformation achieves GDPR compliance, as neither the application nor Edna have the cryptographic keys to read the

sealed data [27], but users can return.

Second, the **account decay** sealing 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 seal ID that can reactivate or completely remove the account after credentials are provided.

Third, we added **category-based content anonymization** so that users can decorrelate their content relating to a particular topic. As per §4.1, this seals 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.

Lobsters's Ruby code communicates with Edna through a JSON-HTTP API. Lobsters sealing transformations are written in 518 LoC of JSON. We added 179 LoC of Ruby to the 160k-LoC Lobsters implementation. We use Lobsters' cronjob and emailing framework to run account decay and send emails. These modifications took less than one person-day for developers familiar with Edna and Lobsters.

6.2 WebSubmit

We integrated Edna as a Rust library with WebSubmit [3]. WebSubmit is a homework submission application used at a U.S. university. 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 data forever. We add support for two sealing transformations: GDPR-compliant user account removal, which removes accounts, and instructor-initiated answer anonymization, which decorrelates student answers for a given course. These transformations allow instructors to retain FERPA-compliant [39] answers after the class has finished. With Edna, students can view and edit their answers after even after class anonymization.

The original WebSubmit has 908 LoC; adding Edna's sealing transformations required an additional 75 LoC to write seal specifications and 312 LoC to implement sealing and revealing HTTP endpoints and modify existing code with e.g., authorization checks for anonymous users. The process took one person-day for a developer familiar with Edna, but unfamiliar with WebSubmit.

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 [19]. HotCRP currently retains past conference data forever and requires manual requests for account removal [20]. We wrote two seal specifications (357 LoC) for HotCRP.

Conference anonymization is invoked by PC chairs after the conference and decorrelates users from their submissions, reviews, and comments, as well as 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 and 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). This account removal sealing transformation removes a user's relationship to co-authored papers, but does not remove the papers themselves. A different policy might go even further and remove papers whose final author is removed.

7 Evaluation

Our evaluation seeks to answer five questions:

1. Does Edna meet its security goals and provide meaningful guarantees to users? (§7.1)
2. Can a policy-enforcement system like Qapla [25] be used to realize Edna-like functionality? (§7.2)
3. How expensive are common application operations, as well as sealing, revealing, and operations over sealed data with Edna? (§7.3)
4. What is the cost of providing stronger guarantees by combining Edna with an encrypted database? (§7.4)
5. What overheads does Edna impose, and where do they come from? (§7.5)

All benchmarks run in a Docker Container running Ubuntu 22.04.1 LTS atop a 40-core server with Intel Xeon E5-2660 v3 CPUs and 64 GB RAM running Arch Linux 5.15.7. We use the MariaDB RDMS with the InnoDB storage engine for application data storage, but store all databases on `tmpfs` to avoid confounding factors related to persistence.

7.1 Security Evaluation

Under Edna's threat model, an attacker can observe all application database content, as well Edna's seal, principal, and deleted principal tables. The attacker can thus learn:

1. any unsealed data in the application database;
2. the active principals that have sealed data, via Edna's principal table
3. the pseudoprincipals currently registered, from Edna's principal table and the application DB;
4. the amount of sealed data in Edna;

5. the seal specifications, from application code.

Importantly, however, Edna at all times protects the confidentiality of the contents of sealed data (via encryption), and hides the existence of removed principals (via deleting their row in the principal table and opaque indexing).

While Edna's sealing reduces plaintext, unsealed information in the application database, remnants of user data may remain unsealed. From looking at the unsealed contents of the application database, an attacker learns:

- nothing about objects' contents if sealing *removes* them;
- the contents of any unmodified object columns, but nothing about the original contents of *modified* columns;
- the existence of objects and their associated pseudoprincipals, but nothing linking pseudoprincipals to a user, provided that sealing *decorrelates* such that each object is owned by a unique pseudoprincipal.

If a developer chooses to decorrelate multiple objects to a single pseudoprincipal, an attacker can learn their grouping and may use this information to infer which principal these objects originally belonged to. Edna relies on developers to choose an appropriate granularity of decorrelation for their desired anonymity guarantees.

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. By induction, the attacker cannot access any private keys.

Edna does not protect against statistical inference attacks based on the size of sealed data, which principals have sealed data, or the number of pseudoprincipals. Viable protections exist, but require expensive padding of sealed data. Edna also makes no guarantees for users who actively use sealed 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 inactive users' sealed data.

Security of Reveal Credentials. If an application chooses to use password-based reveal credentials, Edna guarantees security equivalent to the security of the user's password. Edna cannot reconstruct the private key without either the backup share or the other half of the password-derived share, and Edna cannot brute force the value of the backup share. PBKDF ensures that Edna cannot easily guess the password-derived value with dictionary and rainbow table attacks [43].

Security of Edna-CryptDB. As Edna-CryptDB executes vanilla Edna over an encrypted database, Edna-CryptDB provides the same guarantees as Edna regarding confidentiality of the contents of sealed data. In addition, Edna-CryptDB protects of the confidentiality of the contents of *unsealed* data in the application database against compromise of the database server, subject to the well-known limitations of encrypted databases [16]. An attacker sees only encrypted data, as every query goes through Edna-CryptDB's proxy, and data stored/retrieved is encrypted/decrypted by the owning users' keys. Like CryptDB, Edna-CryptDB does not protect a user u's unsealed data if the attacker (1) compromises a user authorized to view u's data (e.g., an admin); or (2) compromises the application or proxy while u is logged in.

7.2 Comparison With Qapla

We investigate whether a new system (viz., Edna) is indeed required by trying to use Qapla [25] to realize sealing and revealing. We compare the required developer effort, resulting application complexity, and performance to Edna's. Qapla enforces access control policies by rewriting every SQL query with session-specific predicates that restrict the rows returned.

Data removal. In Qapla, a developer realizes sealing transformations that remove data by toggling a metadata flag they add to the schema (e.g., `is_deleted`) from application code. They then provision Qapla with a predicate that checks if this metadata flag is `true` before returning a row. Supporting this metadata flag requires schema and application code changes. This approach cannot realize GDPR-compliant data removal, since the data is still in the database.

Data modification. Qapla only rewrites queries; it does not modify the database. For sealing transformation that modify data, the application developer can use Qapla's "cell blinding" mode. In cell blinding mode, Qapla changes column values based on a predicate before returning query results. Qapla's cell blinding is limited to fixed values and is dynamically applied every time a query runs. By contrast, Edna applies modifications once and makes them stateful in the database.

Data decorrelation. Since Qapla is stateless and does not modify database contents, the developer must manually implement decorrelation by writing application code to create pseudoprincipals and rewrite foreign keys. Qapla can, however, help in enforcing speaks-for permissions (so a user can access data of pseudoprincipals they speak-for) using speaks-for metadata added to pseudoprincipal objects. To do so, developers add a column with the owning principal's ID to the `users` table and have Qapla's predicate check it.

Complexity. Developers must carefully craft Qapla's predicates, which grow in complexity with the number of sealing transformations that can compose. For example, an application supporting both account removal and account anonymization must combine predicates for both removed and owned-by status such that removal always takes precedence. Each additional transformation thus increases the number of predicate clauses whose combinations the developer must reason about.

Qapla's approach does offer some advantages. For example, revealing removed data simply requires toggling a metadata flag (as the data always remains in the database). Revealing also integrates nicely with schema and other application database updates, since the data to reveal remains in the database and adapts according to changes to the database.

Case Study. We implement WebSubmit account removal and answer anonymization (*i*) manually as an irreversible database change; (*ii*) in Qapla (Qapla-WebSubmit); and (*iii*)

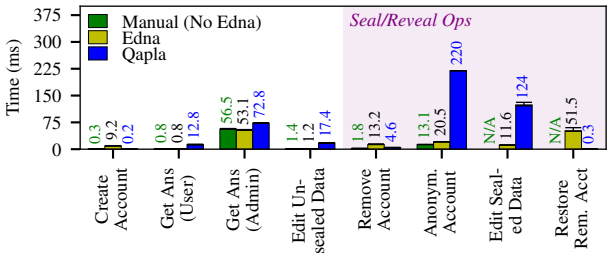


Figure 7: Edna achieves competitive performance with a manual baseline and outperforms Qapla on nearly all operations in WebSubmit (2k users, 80 answers/user). Bars show medians, error bars are 5th/95th percentile latencies.

with Edna (Edna-WebSubmit). All setups require about 300 lines of Rust code to add HTTP endpoints for sealing to WebSubmit. Qapla-WebSubmit requires 576 lines of C/C++ to specify sealing transformations as Qapla policies, and 110 lines of Rust to add pseudoprincipal, modification, and decorrelation support. Edna-WebSubmit requires 75 lines of specification (in JSON), and no extra Rust changes.

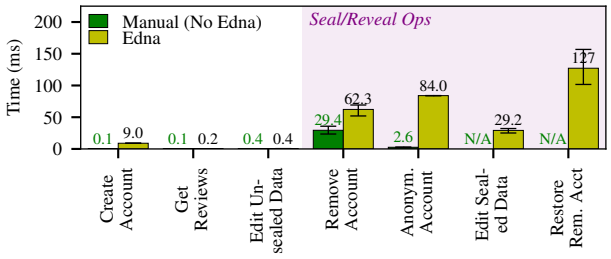
We measure end-to-end latency to perform common application operations (which each issue multiple SQL queries), as well as sealing and revealing operations, in the three setups. Latency includes request processing in WebSubmit, Edna's or Qapla's operations, and the response to the client. We run WebSubmit with a database of 2k users, 20 lectures with four questions each, and an answer for each question for each user (160k total answers). A good result for Edna would show competitive performance with the manual baseline and with Qapla, as Edna does strictly more work (e.g., encrypting sealed data).

Figure 7 shows the results. Qapla-WebSubmit performs well on operations that require only writes since Qapla does not rewrite these 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-WebSubmit performs poorly on operations that require reads, such as listing answers, editing unsealed data, anonymizing accounts, and editing sealed data. Qapla's query rewriting itself takes about 1ms on average, and rewrites `SELECT` queries in ways that affect performance (e.g., requiring additional joins to evaluate predicates).

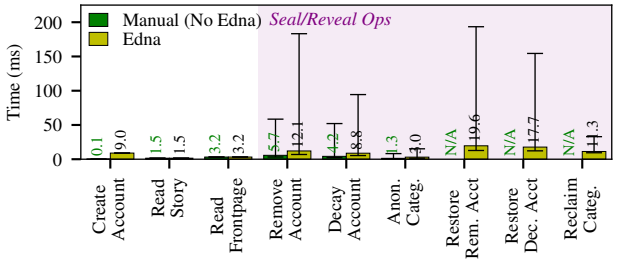
These results demonstrate that Edna achieves better performance than Qapla on common operations. Developers using Qapla must also carefully craft and optimize Qapla predicates (e.g., reducing joins, adding schema indexes and index hints), manually implement some parts of sealing transformations (e.g., decorrelation), and manage schema metadata columns.

7.3 Performance of Edna Operations

We measure Edna's performance in WebSubmit, HotCRP, and Lobsters (§6). As before, we implement a manual, irreversible version of each sealing transformation that directly modifies



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



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

Figure 8: Edna adds no latency overhead to common application operations and modestly increases the latencies of sealing operations compared to a manual implementation. Bars show medians, error bars are 5th/95th percentile latencies.

the database. We measure the latency of common operations, sealing transformations, and Edna-enabled operations over sealed data (e.g., account restoration and editing sealed data). A good result for Edna would show no overhead on common operations, competitive performance with manual sealing, and reasonable latencies for revealing operations impossible in the baseline (e.g., a few seconds for account restoration).

WebSubmit. Figure 7 already compared Edna's operation latency for WebSubmit to the manual baseline. Common operations have comparable latencies with and without Edna; sealing and revealing operations take longer in Edna (11–52ms), but allow users to reveal 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 PC members). HotCRP supports the same sealing transformations as WebSubmit, but PC users have more data (200–300 records each), and HotCRP's sealing transformations mix deletions and decorrelations across 12 tables.

Figure 8a shows higher latencies in general, even for the manual baseline, which reflects the more complex sealing transformations. Edna takes 62.3–127ms to seal and reveal a PC user'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

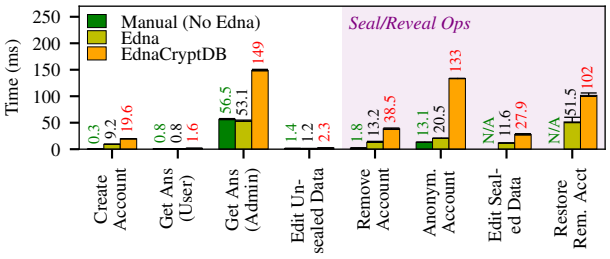
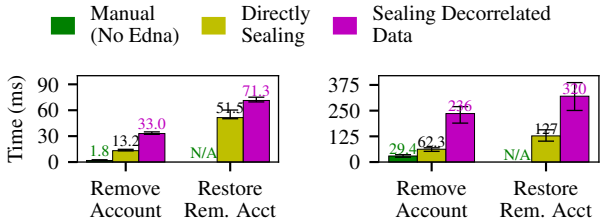


Figure 9: Latencies of WebSubmit (2k users, 80 answers/user) operations when implemented with Edna-CryptDB (adding encrypted database support). Each bar shows the median latency; ranges indicate the 5th to 95th percentile latencies.



(a) WebSubmit (b) HotCRP

Figure 10: Applying sealing transformations to previously-decorrelated accounts increases latency proportionally to the amount of data touched, as Edna performs cryptographic operations linear in the number of pseudoprincipals involved.

proportional to the PC size. With 80 PC members, this transformation takes 6.7s, which is acceptable for a one-off operation. As before, Edna adds minimal latency to common application operations, and 9ms to account creation.

Lobsters. Finally, 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 [23]. Content is distributed among users in a Zipf-like distribution according to statistics from the actual Lobsters deployment [17], and 20% of each user's contributions are associated with the category to anonymize. The benchmark measures server-side latency of common operations and sealing/revealing transformations.

The results are in Figure 8b. The median latencies for entire-account removal or decay are small (8.8–12.1ms for Edna, and 4.2–5.7ms for the baseline), since the median Lobsters user has little data. Revealing sealed accounts takes 17.7–19.6ms in the median. Highly active users with lots of data raise the 95th percentile latency to 80–180ms for sealing and 150–190ms for revealing. Category-based content anonymization touches less data and is faster than whole-account transformations (3.0ms and 11.3ms for the median user to seal and reveal, respectively).

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

7.4 Edna-CryptDB Performance

We now evaluate the cost of adding an encrypted database to Edna. We measure the latency of WebSubmit operations like before, and compare a manual baseline, Edna, and Edna-CryptDB. A good result for Edna-CryptDB would show moderate overheads over Edna, and acceptable absolute latencies.

Figure 9 shows the results. Normal application operations are 2–3× slower than in Edna, with operations that access many rows, such as the admin viewing all answers, most affected. Sealing and revealing operations are also 2–6.5× 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 to queries; a no-op version of our proxy makes operations 1.03-1.5× slower. Cryptographic operations themselves are cheap (< 0.2ms to generate keys or encrypt/decrypt). 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, expensive queries. This is particularly expensive when the user owns many keys (e.g., the WebSubmit admin). Admin-applied anonymization incurs the highest overhead (+119.9ms) 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 (+92.5ms).

7.5 Edna Performance Drill-Down

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 cryptographic operations are comparatively expensive. PBKDF2 hashing for private key management incurs a 9ms cost and affects account registration and operations on sealed data that reconstruct a user's private key. Encryption and decryption incur baseline costs of 0.1ms and 0.02ms respectively; cost grows linearly with data size. In the common case, sealing or revealing data performs two cryptographic operations (one to encrypt/decrypt the seal records, and one to encrypt/decrypt the ID at which they are stored). When the application applies multiple sealing transformations and seals the data of pseudoprincipals, several encryptions/decryptions may be required; we evaluate this cost next.

7.5.1 Composing Sealing Transformations

We measure the cost of composing account removal on top of a prior sealing 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 sealing transformation; and (*iii*) account removal and restoration *with* a prior anonymization sealing transformation. In the final case, a subset of the user's data has already been decorrelated when removal occurs, and removal thus encrypts sealed data with pseudoprincipals' public keys. Restoring the removed, anonymized account must then individually decrypt pseudoprincipal records. Hence, sealing and revealing in the third setup should take time proportional to the number of pseudoprincipals created by anonymization.

Figure 10 shows the resulting latencies. WebSubmit account removal and restoration latencies both increase by 19.8ms; HotCRP removal latency increases by 173.7ms and restoration by 193ms. Accounts in HotCRP have more data and 14–15× more pseudoprincipals after anonymization than those in WebSubmit, so these slowdowns make sense.

Importantly, sealing latencies stabilize when Edna composes further sealing 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).

7.5.2 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 seal record storage uses 12MB, which grows to 58.5MB after the users remove their accounts, The database also increases from 261MB to 290MB. (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 (another 2.2MB).

Like all encrypted databases, 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.

7.5.3 Impact On Concurrent Application Use

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

We thus measure the impact of Edna's operations on other concurrent requests in Lobsters. In the experiment, a set of "normal" users make continuous requests to the application

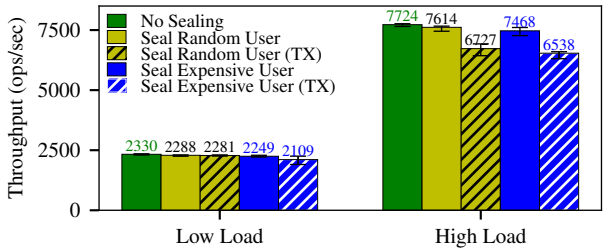


Figure 11: Even continuous sealing/revealing operations by a heavy-hitter user in Lobsters (> 6k data objects) have a small impact on application request throughput.

while another distinct set of users continuously remove and restore their accounts. Edna applies sealing transformations sequentially, so only one transformation happens at a time. We measure the throughput of "normal" users' application operations in closed-loop setting, 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 [17].

Since users' sealing/revealing costs vary in Lobsters, we measure the impact of (1) randomly chosen users invoking account removal/restoration, and (2) 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 (≈98% CPU load). Finally, we measure settings with and without a transaction for Edna transformations. A good result for Edna would show little impact on the throughput of normal operations when concurrent sealing transformations occur.

Figure 11 shows the results. Under both low and high load, the throughput of normal operations is largely unaffected by concurrent sealing, dropping by at most 3.4% without transactions (and < 15% with transactions). This shows that Edna's sealing and revealing transformations have little impact on other users' application experience. Account removals and restorations under high load take longer, with the expensive user's account removal taking 6.1 seconds and revealing 10.6 seconds. This is acceptable: 50% of data deletions at Facebook take five minutes or longer to complete [12].

8 Conclusion

Edna enables developers to provide data sealing and revealing transformations that give users control over their data in web applications. Sealing balances between users' desire for their data to be protected and applications' need for this data, and protects user data against exposure via common vulnerabilities. We used Edna to add seven sealing transformations to three web applications, and found that Edna's sealing and revealing operations are fast and impose little overhead on normal application operation. We will open-source Edna.

References

[1] URL: https://www.ietf.org/rfc/rfc2898.txt.

[2] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. "Blockstack: A Global Naming and Storage System Secured by Blockchains". In: *Proceedings of the 2016 USENIX Annual Technical Conference (ATC)*. Denver, Colorado, USA, June 2016, pages 181–194.

[3] Anonymous Authors. *WebSubmit: a simple class submission system*. Unblinded citation supplied on HotCRP. URL: https://github.com/XXXXX (visited on 06/03/2020).

[4] 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.

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

[6] 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.

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

[8] 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: *15th Workshop on Hot Topics in Operating Systems (HotOS)*. Kartause Ittingen, Switzerland: USENIX Association, May 2015.

[9] 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.

[10] 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'10. Boston, MA, 2010, page 1.

[11] 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.

[12] 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.

[13] Amol Deshpande. "Sypse: Privacy-first Data Management through Pseudonymization and Partitioning". In: *The Conference on Innovative Data Systems Research (CIDR)*. Chaminade, CA, Jan. 2021.

[14] "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.

[15] 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.

[16] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. "Why Your Encrypted Database Is Not Secure". In: May 2017, pages 162–168.

[17] 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).

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

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

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

[21] 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.

[22] 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.

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

[24] 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 Workshop on Hot Topics in Operating Systems (HotOS)*. 2019, pages 88–95.

[25] 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.

[26] 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.

[27] European Network and Information Security Agency. *Privacy and data protection by design: from policy to engineering*. 2015. URL: https://data.europa.eu/doi/10.2824/38623.

[28] 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.

[29] 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).

[30] 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).

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] Adi Shamir. URL: https://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf (visited on 04/18/2022).

[37] Alexander H. Southwell, Ryan T. Bergsieker, Cassandra L Gaedt-Sheckter, Frances A. Waldmann, and Lisa V. Zivkovic. *Virginia Passes Comprehensive Privacy Law"*. Mar. 2021. URL: https://www.gibsondunn.com/virginia-passes-comprehensive-privacy-law/ (visited on 03/08/2021).

[38] Griffin Thorne. *GDPR Meets its Match ... in China*. July 2019. URL: https://www.chinalawblog.com/2019/07/gdpr-meets-its-match-in-china.html (visited on 06/04/2020).

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

[40] 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.

[41] 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).

[42] 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.

[43] Yin, Yiqun Lisa Yao, Frances F. URL: http://palms.ee.princeton.edu/PALMSopen/yao05design.pdf (visited on 04/18/2022).

[44] Wen Zhang, Eric Sheng, Michael Chang, Aurojit Panda, Mooly Sagiv, and Scott Shenker. "Blockaid: Data Access Policy Enforcement for Web Applications". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, July 2022, pages 701–718.