Privacy Heroes Need Data Disguises

Anonymous Authors

Abstract. Conscientious application developers want to provide better user privacy; legal mandates and user demand increasingly require it. Unfortunately, privacy in complex, data-rich applications is fundamentally hard. Consider a user who wants to remove their account from a service. Even finding the corresponding data is nontrivial, but once it is found, only some of it should be removed; other data should be anonymized or decorrelated (for legal reasons, or to maintain application utility for other users), and some of these transformations should be reversible (in case a user wants to return).

We propose *data disguising*, a systematic approach that helps developers generate privacy transformations for database-backed web applications from a high-level specification and preexisting data relationships. Data disguising simplifies privacy transformations that applications use today, supports fine-grained and nuanced policies that would be cumbersome to implement manually, and enables reversible transformations for users who change their mind. A prototype tool for data disguising demonstrates that our approach is practical.

1 Introduction

Web application developers today have more incentives than ever to provide better privacy for their users. Laws like the EU's General Data Protection Regulation (GDPR) [15] and California's Consumer Privacy Act (CCPA) [6] codify users' right to be forgotten, and restrict any data retention to anonymized information. Legal consequences and the reputational damage associated with data breaches [27, 28, 37, 41, 50] make it good practice to minimize the user data retained at any point.

Although many developers are well-intentioned, today they must implement *privacy transformations* (such as user account deletion or data minimization) manually, which results in coarse-grained transformations and ad-hoc solutions. To get privacy transformations right, developers must carefully map the high-level privacy policy to operations that delete or rewrite data objects, while ensuring that the application preserves utility for other users, retains legally-mandated anonymized data, and avoids violating application invariants. For example, deleting a user's account should not unexpectedly grant broad access to previously-private content by deleting objects that restrict access, nor should it make non-sensitive shared content disappear. Privacy transformation designers must correctly handle identifying data, as well as indirectly identifying correlations between data objects. For example, anonymized public running routes can identify the user's hometown and be reassociated with the user [20], and anonymized order histories in e-commerce sites can reidentify buyers.

The burden of implementing privacy transformations grows with the underlying privacy policy's complexity. But though simplicity has advantages, more nuanced privacy policies are important and useful. Such policies can help protect users against data correlation attacks; they can give more control to individuals by allowing them to choose their own, fine-grained privacy semantics; and they may enable new privacy modes such as data that gradually "decays" to become less identifiable over time. Likewise, reversible privacy transformations might strike a sweet spot—allowing users to remove identifiable information on demand, but accommodating service providers' interest to make it easy for users to return—but adds significant burden to implementation.

We propose *data disguising*, a new framework for systematically specifying and implementing privacy transformations. In this framework, developers specify transformations required in privacy policies as high-level *data disguises* over existing application data types and associations. Applying a disguise transforms the state of application data in privacy-preserving ways (e.g., deleting users' identifiers, or decorrelating identifying object relationships) while preserving application invariants and utility. Disguises do not guarantee strict privacy—a disguise's content might still expose user information or correlated data if not redacted by the policy—but flexible disguising,

rather than universal deletion, is what real applications require. A data disguising tool takes a disguise and its target, and automatically applies the appropriate transformations to application data to achieve the disguised state, reducing the developer burden. Our proof-of-concept, Edna, demonstrates the potential of such a tool. Edna proxies relational database operations and exposes APIs to invoke privacy transformations.

2 The Need for Privacy Transformations

We surveyed the account deletion behavior of several widely-used web applications to understand what privacy transformations they apply. A set of common themes emerged, as well as directions for future improvement.

Treatment of user contributions. The applications surveyed delete some user profile information (e.g., passwords) from underlying databases, though all applications surveyed retain some information for legal or necessary business purposes (e.g., Spotify fraud detection [46], PrestaShop/Amazon orders [3, 21]). The treatment of other user contributions varies. Some services keep most contributions publicly and indefinitely available (e.g., StackOverflow answers [25], shared Strava routes [47]), sometimes associated with the original user name (e.g., Wikipedia edit history [16]), even if a user deletes their account. Social networking platforms delete some of a user's contributions (e.g., posts), but keep others unanonymized and visible to their recipients (e.g., Facebook/Twitter private messages [22, 26], LinkedIn updates [32]). Other platforms with mostly public content keep user contributions visible to the intended audience, but anonymize them by reattributing the contribution to a placeholder user (e.g., GitHub's "@ghost" [23], Reddit and Lobsters' "[deleted]" [24, 34]).

In the open-source applications surveyed, developers implement privacy transformations via ad-hoc database operations, and only trigger them on explicit, user-initiated account deletion. Most developers appear to pay little attention to identifying correlations within the remaining data.

Nuanced policies. Users and application developers can both benefit from more nuanced privacy policies. For example, a confidential paper review system like HotCRP [19] must keep a user's contributions (papers, reviews) after they delete their account to preserve utility for others, but could associate each review with a different placeholder to avoid revealing the deleted reviewer's identity. In some cases, a policy could preserve utility while reducing the efficacy of later inference attacks by changing posts, such as by modifying their metadata (e.g., tags, posting times). Some transformations should be performed only on sensitive metadata: a tag like "#cat" is likely insensitive and useful to preserve, whereas one naming a person is not. Individual users may even specify different preferences for their data.

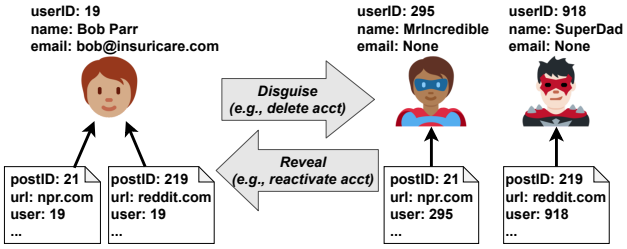

Figure 1: Disguises move the target object (in this example, a user Bob) from an identity-revealing guise to privacy-preserving guises.

Data decay. Applications could go beyond simple account deletion and support a data expiration policy that anonymizes a user's contributions after the user has been inactive for a period of time, possibly restoring the user's profile and contributions if the user ever logs back in. Or the application could gradually "decay" sensitive data by applying several privacy transformations that incrementally remove identifiable information over time.

Reversibility. Many applications might wish to employ *reversible* transformations to, for example, support account reactivation instead of permanent and irrevocable account deletion. After all, if services must allow users to remove their data on request, it is in the operator's interest as well as the user's to make it easy for a user to change their mind and return, bringing their data along. An advanced reversible transformation might record all data transformed, and push an encrypted archive to third-party cloud storage. If the user wishes to return, they supply the archive to reverse the transformation. To ensure access even if the user loses their key, the transformation might secret-share the encryption key [45] among the user, the service, and a trusted third party (e.g., the ACLU or EFF).

Privacy transformations. Taken together, these properties argue for a systematic treatment of privacy transformations that makes them a first-class citizen in application design. In particular, we imagine that developers declaratively specify the above and other transformation policies like they specify a storage structure (e.g., a relational schema) today. This also allows for new policies and use cases that benefit both end-users and service operators.

3 Data Disguising

The key idea behind *data disguising* is to associate multiple *guises* with a target data object. Guises vary in how they reveal identities or preserve privacy. Objects move between different guises by means of privacy transformations. Figure 1 illustrates this with the example of user account deletion. When his account is active, user

User Transformation Spec

```
"id":       IDAttribute,
"name":     Gen(Random),
"active":   Gen(Default(false)),
"darkmode": CopyAll,
"notifs":   CopyOnce+Gen(Default(false)),
"tag_id":   GenForeignKey,
```

User Object

```
"id":       19,
"name":     BobParr,
"active":   true,
"darkmode": false,
"notifs":   true,
"tag_id":   11
```

Guise 1

```
"id":       295,
"name":     MrIncredible,
"active":   false,
"darkmode": false,
"notifs":   true,
"tag_id":   81483
```

Guise 2

```
"id":       918,
"name":     SuperDad,
"active":   false,
"darkmode": false,
"notifs":   false,
"tag_id":   15592
```

Figure 2: Creating two guises of an example user (of a synthetic application schema).

Bob's profile is associated with his true identity and all his contributions to the site (an identity-revealing guise). When Bob deletes his account, his profile and contributions move to different, privacy-preserving guises: his name has been anonymized, his email address has been redacted, and his contributions have been decorrelated and attributed to individual, unidentified user guises.

Data disguising builds on the existing structure of web applications. Web applications are often structured as object graphs, either explicitly [4, 11], through an object-relational model (ORM) [48], or implicitly via foreign keys (edges) between tables (vertices) in relational databases. Data disguises transform this object graph.

The application developer writes a disguise specification for each privacy transformation needed in the application. This specification is a declarative statement similar to a relational schema, with entries for graph vertices (objects) and directed edges (relationships between pairs of objects) to be transformed (see §4). We assume that:

1. developers use their domain knowledge to write correct and complete disguises;
2. application code handles the different guises appropriately (e.g., in displaying them); and
3. different guises of the same object have the same structure (e.g., they can be rows in the same table).

A data disguising tool takes the disguise specification and turns it into storage operations that apply the transformation (disguise) or its reverse (reveal).

At any given moment, an application's data object graph comprises a mix of identity-revealing guises and privacy-preserving ones. Privacy transformations split and combine individual guises when triggered.

4 Specifying Data Disguises

Data disguises involve transformations on object types and edge types in the application object graph. A disguising tool applies these transformations by traversing the graph starting from a developer-specified target object, such as a deleted user.

4.1 Creating Guises for Objects

To create a guise from an object, developers specify how to transform attributes of the object into guise attributes. Figure 2 shows an example, producing guises for user objects. User objects have identifier `id`; a reference `tag_id` forms an edge in the graph (a foreign key constraint to tag objects).

Guises always have unique, random identifiers. Developers choose how to create other guise attributes, selecting from among the following:

(1) Copy object content. Guises of the same object all share the object's attribute values. If the attribute is an edge attribute (e.g., a foreign key column), all guises will have edges to the same object. Copying allows developers to retain the object's content, without worrying about how to synthesize attribute values for guises. For example, in Figure 2 the `darkmode` attribute is copied in all guises.

(2) Generate new content. To create new attributes, developers specify whether the guise's value should be random, a default value, or generated from the object's attribute value via a custom function (e.g., hashing the value). Figure 2 illustrates an example of random (`name`) and default (`active`) generated value attributes. Creating new guise edge attributes (e.g., new foreign key relationships) requires creating a new guise for the referenced object in order to maintain referential integrity; the data disguise rewrites the edge to point to the new guise. In Figure 2, creating two user guises requires creating two tag guises, and the tag guises' identifiers become the user guises' foreign keys.

(3) Copy object content, but only once. One guise copies the attribute value from the object, but all other guises generate new values (as described above). `notifs` in Figure 2 illustrates how the attribute is copied once. This enables the application to retain the original object semantics (e.g., a count of how many users want notifications) without creating duplicates.

4.2 Transforming Edges Between Objects

Edge transformations specify how a disguise restructures the object graph as it recursively traverses edges from the target. Edges have a *source* object and a *destination* object, where the source references the destination (e.g., via a foreign key column in a relational database). The developer specifies transformations for each edge type, choosing from the following:

1. **Retain** edges of this type. Transforms both source and destination objects into a single guise each.
2. **Decorrelate** edges of this type. Creates one guise of the destination object for each of the sources; replaces each source's edge to the destination with an

edge to that unique guise.

3. **Delete** edges of this type. Transforms the destination object into a single guise; removes the source and its descendants if otherwise unconnected.

An optional argument enables conditionally-applied edge transformations, e.g., removing only user-created tags, or removing tags only until the user's papers comprise less than 10% of tagged papers.

4.3 An Example Disguise

Consider disguising Bob when he deletes his HotCRP account. Bob would prefer his papers and reviews to be unlinked from his identity. HotCRP, on the other hand, would like to retain paper and review information that other users find useful. A careful selection of edge and object transformations achieves both.

To decorrelate reviews from Bob, the disguise `Decorrelates` user-to-review edges. This requires transforming Bob into one unique user guise per review. The disguise generates guise attribute values using suitable defaults; in particular, HotCRP users' `disabled` attribute is set for the guises, ensuring that guises have no permissions and never review papers.

Bob is further linked to papers through conflicts, which can indicate coauthorship or a reviewer conflict. These conflicts are not reassigned to the new guises, since preserved conflicts could reidentify Bob as the likely author of a review. Thus, the conflict edges that link a disguised user to papers need a `Delete` annotation.

The disguise `Retains` all other edge types, ensuring that review and paper artifacts remain correctly linked. Active reviewers still see the correct paper for their reviews, and active authors see the correct reviews for their papers, albeit potentially authored by anonymous, unlinkable guises of the original reviewer.

Unlike the current real-world HotCRP account deletion policy [29], which deletes all objects belonging to Bob, this disguise strikes a balance between decorrelating Bob's identity from his reviews and papers, and maintaining useful information for other HotCRP users. Furthermore, it is easy to imagine extending this disguise to automatically disguise Bob after some time (e.g., 2 years after the conference), protecting his future research career by hiding youthful reviewing sins.

5 Prototype

Our prototype disguising tool, Edna, is a MySQL proxy consisting of 5K LoC of Rust. It supports standard SQL as well as `DISGUISE [disguise_id] [target]` and `REVERSE DISGUISE [disguise_id] [target]` queries, where `disguise_id` identifies a declarative disguise specification and `target` is the unique ID of the target object. Edna provides data disguising for applications that use relational databases, where objects are table rows and edges are foreign key relationships. We believe that the data disguising approach can be extended to other types of stores.

Application Disguise	#Object Types	#Edge Types	Schema LoC	Disguise LoC
Lobsters-Current	19	20	318	220
HotCRP-Current	25	50	352	378
HotCRP-Granular	25	50	352	407

Table 1: Data disguise specifications for Lobsters and HotCRP have similar complexity to a relational schema.

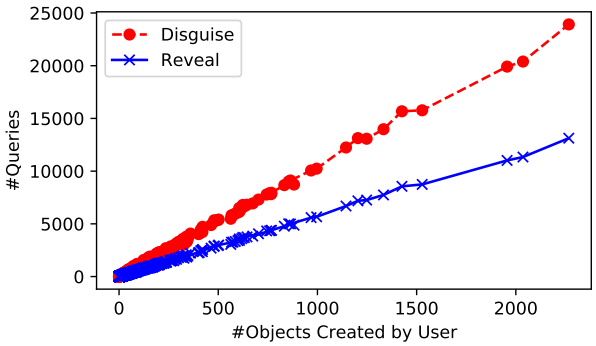

Figure 3: Number of queries required by Lobsters-Current to disguise and reveal a target user, depending on the number of objects created by the user.

6 Case Studies

We evaluate whether Edna enables developers to easily express and automate practical data disguises by writing disguises for user account deletion in Lobsters [33], an open-source news feed application, and HotCRP [19]. We consider three settings: Lobsters-Current and HotCRP-Current implement the current account deletion policies in the two applications [29, 34]. HotCRP-Granular specifies a HotCRP account deletion policy that balances useful data retention with data deletion for privacy (§4.3).

Complexity. We would hope that writing disguises involves similar labor and difficulty as writing relational schemas. In particular, a developer should write a disguise only once. Because disguises require at most one transformation for each object type and edge type, disguise complexity is limited by the number of object types and relations in the application schema. Table 1 shows that the disguise specification for our applications is indeed comparable in size to the applications' schemas.

Performance. Our experience implementing Edna exposed the challenging performance tradeoff between efficiently executing normal application queries, and achieving low-latency disguise operation. The right balance depends on the rate of disguising, which may range from rare (as in today's applications) to quite frequent (in a privacy-supporting world where users freely disguise and

reveal themselves, or where data automatically ages out).

One Edna variant intercepts application queries, which decreases throughput in normal execution by 4% compared to a baseline without Edna. Another variant additionally tracks an in-memory object graph, decreasing throughput by 9% compared to the baseline. When executing a privacy transformation, the worst case generates as many guises as there are edges connected to the target, each requiring one or more SQL queries to create. Figure 3 shows the number of queries run when applying and reversing Lobsters-Current on an object graph with 3K users and 80K total user contributions. The overhead increases linearly with the number of objects created by the target user. The reverse transformation (reveal) starts with the complete data and batches by table, while disguising currently applies queries as it traverses the object graph; some batching is possible and could improve this.

Our results indicate the need for privacy-aware database designs, which spread the cost of disguising across normal execution. One possibility keeps a "shadow" copy of the database that reflects the disguised state, with guises already created and edges rewritten. Pre-created guises are hidden by transparently exposing materialized views (MV) to the application. Disguising then exposes the shadow copy in the MV, and removes disguised objects. This achieves good read performance—queries read from MVs instead of the database—albeit at the cost of space overhead. However, normal execution write queries must potentially update multiple guises in the shadow copy, and pre-created guises consume space during normal execution, even if disguises are never applied. Other possibilities include making disguises eventually consistent; updating the shadow copy asynchronously; and caching disguises for target objects disguised most often.

7 Discussion

Data disguising makes it easier to explore and implement privacy transformations. This may support new privacy paradigms like data decay and account reactivation after deletion. Data disguising also has some clear limitations. It assumes that transformations on the object graph capture all data that needs transforming; application snapshots, backups, or external data copies are out of scope, and require e.g., taint tracking techniques [30]. Data disguising also importantly does not provide privacy *guarantees*. Disguises are only as good as the specification a developer writes for them, and we assume that developers capture application-specific needs to retain, remove, and decorrelate data in their policies. We imagine that data analysis tools and heuristics can help developers improve or catch errors in disguise specifications, similar to techniques used by Facebook to detect incorrect deletion [11]. Users of applications that use data disguises cannot blindly assume that the disguises provide complete privacy, although they can perhaps vet application privacy properties via the disguise specification.

Edna currently works only with relational databases, and does not support dynamic updates to the application schema or disguise specification. Database schema evolution research [12] may offer insights into supporting disguises and disguise reversals after such updates.

8 Related Work

Two prior **systems for personal data deletion** are particularly relevant to data disguising: Sypse [13], which pseudonymizes user data and partitions personally identifying information (PII) from other data, and DELF [11], a framework for data deletion at Facebook. While Sypse also relies on an object graph, its design is application-oblivious, and leaves the specification of what counts as PII as future work. Data disguising provides a way to specify sensitive data and correlations, as well as application-specific transformations. Instead of constantly maintaining data partitions (as Sypse does), data disguising adapts a user's data when these transformations are invoked. DELF [11] helps developers write correct data deletion code, and uses annotations on application edge and object types to specify a deletion policy. DELF focuses only on data deletion, while data disguising targets broader privacy transformations, including decorrelating edges.

Database designs for GDPR compliance [30, 44] track the owner of data objects and erase them when requested under the GDPR. They either modify the data layout [44] or use fine-grained information flow tracking (IFC) to determine PII propagation and restrict access [30]. Data disguising can be employed for GDPR compliance, but supports privacy transformations beyond deleting PII, and requires no ownership tracking or fine-grained IFC.

Other systems enforce developer-specified **visibility and access control policies** based on general information flow control approaches [10, 17, 18, 43, 49], authorized views [5] or per-user views [35], and rewriting database queries [36, 40]. Data disguising transforms the actual data stored according to a specification.

Privacy-preserving data mining approaches, such as k-anonymity, l-diversity, and differential privacy [1, 14], provide **statistical privacy guarantees**. These complement data disguising: disguise decorrelation thresholds might be based on differential privacy, for example.

Finally, **clean-slate designs** for user-centric data ownership paradigms seek to "decentralize" the internet [2, 7–9, 31, 38, 39, 42], granting ultimate control over data to end-users. These systems typically lack the capacity for server-side compute, burden users with long-term data maintenance, and break the current application revenue model. In contrast, data disguising helps developers specify and automate privacy transformations without changing the application data model or business model.

References

[1] Charu C. Aggarwal and Philip S. Yu. "A General Survey of Privacy-Preserving Data Mining Models and Algorithms". In: *Privacy-Preserving Data Mining: Models and Algorithms*. Edited by Charu C. Aggarwal and Philip S. Yu. Boston, MA: Springer US, 2008, pages 11–52.

[2] Muneeb Ali, Ryan Shea, Jude Nelson, and Michael J. Freedman. *Blockstack Technical Whitepaper*. 2017.

[3] Louise Bonnard. *Complying with the GDPR*. May 2018. URL: https://doc.prestashop.com/display/PS17/Complying+with+the+GDPR (visited on 12/17/2020).

[4] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, et al. "TAO: Facebook's distributed data store for the social graph". In: *2013 USENIX Annual Technical Conference (USENIX ATC 13)*. 2013, pages 49–60.

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

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

[7] 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 15)*. Kartause Ittingen, Switzerland: USENIX Association, May 2015.

[8] 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.

[9] 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: USENIX Association, 2010, page 1.

[10] Adam Chlipala. "Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications". In: *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. Jan. 2010, pages 105–118.

[11] 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: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020.

[12] Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. "Automating the database schema evolution process". In: *The VLDB Journal* 22.1 (2013), pages 73–98.

[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] Cynthia Dwork. "Differential privacy: A survey of results". In: *International conference on theory and applications of models of computation*. Springer. 2008, pages 1–19.

[15] "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.

[16] Wikimedia Foundation. *Privacy Policy*. May 2018. URL: https://foundation.wikimedia.org/wiki/Privacy_policy (visited on 01/26/2021).

[17] 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: *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX Association, Oct. 2012, pages 47–60.

[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] Jeremy Hsu. *The Strava Heat Map and the End of Secrets*. Jan. 2018. URL: https://www.wired.com/story/strava-heat-map-military-bases-fitness-trackers-privacy (visited on 02/01/2021).

[21] Amazon.com Inc. *Amazon.com Privacy Notice*. Jan. 2020. URL: https://www.amazon.com/gp/help/customer/display.html?nodeId=GX7NJQ4ZB8MHFRNJ (visited on 12/17/2020).

[22] Facebook Inc. *Data Policy*. Aug. 2020. URL: https://www.facebook.com/about/privacy (visited on 12/17/2020).

[23] GitHub Inc. *GitHub Privacy Statement*. Nov. 2020. URL: https://docs.github.com/en/free-pro-team@latest/github/site-policy/github-privacy-statement (visited on 12/17/2020).

[24] Reddit Inc. *Reddit Privacy Policy*. Sept. 2020. URL: https://www.redditinc.com/policies/privacy-policy (visited on 12/17/2020).

[25] Stack Exchange Inc. *Stack Exchange, Inc. Terms of Service*. Feb. 2020. URL: https://stackoverflow.com/legal/terms-of-service (visited on 01/26/2021).

[26] Twitter Inc. *Twitter Privacy Policy*. June 2020. URL: https://twitter.com/en/privacy (visited on 12/17/2020).

[27] Mike Isaac and Sheera Frenkel. *Facebook Security Breach Exposes Accounts of 50 Million Users*. Sept. 2018. URL: https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html (visited on 12/19/2020).

[28] Rebecca Klar. *Twitter becomes first US tech firm fined for EU privacy law violation*. Dec. 2020. URL: https://thehill.com/policy/technology/530238-twitter-becomes-first-us-tech-firm-fined-for-eu-privacy-law-violation (visited on 12/19/2020).

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

[30] 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*. Edited by Vijay Gadepally, Timothy Mattson, Michael Stonebraker, Fusheng Wang, Gang Luo, Yanhui Laing, and Alevtina Dubovitskaya. Springer, 2019, pages 24–38.

[31] Maxwell Krohn, Alex Yip, Micah Brodsky, Robert Morris, and Michael Walfish. "A World Wide Web Without Walls". In: *6th ACM Workshop on Hot Topics in Networking (HotNets 07)*. Atlanta, GA, Nov. 2007.

[32] LinkedIn. *LinkedIn Privacy Policy*. Jan. 2021. URL: https://www.linkedin.com/legal/privacy-policy (visited on 01/26/2021).

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

[34] Lobsters. *Privacy Policy*. URL: https://lobste.rs/privacy (visited on 12/17/2020).

[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 Workshop on Hot Topics in Operating Systems (HotOS 19)*. 2019, pages 88–95.

[36] Aastha Mehta, Eslam Elnikety, Katura Harvey, Deepak Garg, and Peter Druschel. "Qapla: Policy Compliance for Database-Backed Systems". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pages 1463–1479.

[37] Eliabeth Montalbano. *Data from August Breach of Amazon Partner Juspay Dumped Online*. Jan. 2021. URL: https://threatpost.com/data-from-august-breach-of-amazon-partner-juspay-dumped-online/162740 (visited on 01/06/2021).

[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] Shoumik Palkar and Matei Zaharia. "DIY Hosting for Online Privacy". In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets 17)*. 2017, pages 1–7.

[40] 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).

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

[42] 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.

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

[44] 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.

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

[46] Spotify. *Spotify Privacy Policy*. Jan. 2020. URL: https://www.spotify.com/us/legal/privacy-policy (visited on 12/17/2020).

[47] Strava. *Strava Privacy Policy*. Dec. 2020. URL: https://www.strava.com/legal/privacy (visited on 12/17/2020).

[48] Alexandre Torres, Renata Galante, Marcelo S. Pimenta, and Alexandre Jonatan B. Martins. "Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design". In: *Information and Software Technology* 82 (2017), pages 1–18.

[49] 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.

[50] Raymond Zhong. *Quora, the Q. and A. Site, Says Data Breach Affected 100 Million Users*. Dec. 2018. URL: https://www.nytimes.com/2018/12/04/technology/quora-hack-data-breach.html (visited on 12/19/2020).