Position: GDPR Compliance by Construction

Malte Schwarzkopf[1], Eddie Kohler[2], M. Frans Kaashoek[1], and Robert Morris[1]

[1] MIT CSAIL
[2] Harvard University

Abstract. New laws such as the European Union's General Data Protection Regulation (GDPR) grant users unprecedented control over personal data stored and processed by businesses. Compliance can require expensive manual labor or retrofitting of existing systems, *e.g.*, to handle data retrieval and removal requests. We argue for treating these new requirements as an opportunity for new system designs. These designs should make data ownership a first-class concern and achieve compliance with privacy legislation by construction. A *compliant-by-construction* system could build a shared database, with similar performance as current systems, from personal databases that let users contribute, audit, retrieve, and remove their personal data through easy-to-understand APIs. Realizing compliant-by-construction systems requires new cross-cutting abstractions that make data dependencies explicit and that augment classic data processing pipelines with ownership information.

We suggest what such abstractions might look like, and highlight existing technologies that we believe make compliant-by-construction systems feasible today. We believe that progress towards such systems is at hand, and highlight challenges for researchers to address to make them a reality.

1 Introduction

Many websites store and process customers' personal data in server-side systems. Companies operating these websites must comply with data protection laws and regulations, such as the EU's General Data Protection Regulation (GDPR) [9] and the California Consumer Privacy Act of 2018 [1], that grant individuals significant control of and powers regarding their own data. For example, the GDPR makes it mandatory for enterprises to promptly provide users with electronic copies of their personal data ("right of access"), and for enterprises to completely remove the user's personal data from its databases on request ("right of erasure"). Non-compliance with the GDPR can result in severe fines of up to 4% of annual turnover, and the EU has recently imposed fines of hundreds of millions of dollars on Mariott [28] and British Airways [27] for negligent handling of customer data. At one estimate, the cost of GDPR compliance is expected to exceed $7.8bn for U.S. businesses alone [25].

In this paper, we survey the challenges that GDPR compliance creates for data storage and processing systems, and argue that the database and systems research communities ought to treat these challenges as an opportunity for new system designs. These new designs should broadly treat end-users' control over their personal information as a first-class design concern. Our vision is to align system designs closely with the reality of data ownership and legislative requirements such as those imposed by the GDPR. Concretely, systems should achieve compliance with GDPR-like legislation *by construction*, with significantly more help from databases than current systems can offer.

The GDPR differs from prior privacy legislation primarily in its comprehensiveness. The GDPR has an expansive interpretation of "personal data" that covers any information related to an identifiable natural person, the *data subject*. The legislation establishes data subjects' rights over the information that *data controllers* (*e.g.*, web services) collect and which *data processors* (*e.g.*, cloud providers) store and process.

This has wide-ranging implications, including for the relational backend databases that support web applications. To comply with the spirit of individual control over data and to guarantee the data subjects' rights, such databases should become *dynamic, temporally-changing federations of end-users' contributed data*, rather than one-way data ingestors and long-term storage repositories that indiscriminately mix different users' data.

To realize this ideal, a database must:

1. logically separate users' data, so that the association of ingested, unrefined "base" records with a data subject remains unambiguous;
2. model the fine-grained dependencies between derived records and the underlying base records; and
3. by appropriately adapting derived records, handle the removal of one user's data without breaking high-level application semantics.

More ambitious goals may include having the database attest the correctness of its ownership tracking and data removal procedures, or to synthesize such procedures from a high-level privacy policy.

Today's websites and applications rely on much more than databases, however: blob stores may store artifacts like photos, machine learning models may train on users' derived data, and long-term analytics pipelines may update aggregate statistics on dashboards. Consequently, implementing our vision in any single system is likely insufficient. Instead, *cross-cutting* abstractions that generalize across systems are needed.

We believe that one promising approach is to conceptualize web service backends—databases, blob stores, analytics pipelines, machine learning (ML) models, and other systems—as a large *dataflow computation*. In this model, a user "subscribes" and contributes her data into an exclusively-owned shard of the backend. This shard stores all data owned by this user. As data arrives into the shard, it streams into other systems for processing and storage, but remains associated with the contributing user. The user may at any point choose to withdraw her shard (and thus, her data) from the system. For example, a newly-uploaded picture will initially be associated with the uploader's shard (and logically, or perhaps even physically, stored with it). Subsequently, it may percolate into a blob store (for storing the binary image data), a database (for tracking the picture's metadata), a request log, and a notification service that pushes updates to the uploader's friends. Beyond the original, encrypted user shard, the picture is associated only with a pseudonymous identifier, a model that is GDPR-compliant by construction. If the uploading user decides to retrieve her data, the service merely needs to return her shard and all information it contains; if she demands erasure of her data, the service deletes the shard and streams revocation messages that remove derived data.

The dataflow architecture we sketched here crucially requires systems to track data origin, *e.g.*, via explicit labels or well-defined relationships. In addition, all systems must support both data contribution and data revocation. We believe that efficient mechanisms for these purposes are within reach, and that developing mechanisms and systems that achieve compliance by construction constitutes a fruitful research direction for databases and privacy-aware and distributed datacenter systems.

2 Vision

We envision a compliant-by-construction web service backend that allows users to seamlessly introduce, retrieve, and remove their personal data without manual labor on the application developer's part. In the following, we focus how this vision addresses data subjects' rights to access, objection, erasure, and data portability under the GDPR. Existing techniques discussed in §4 are sufficient to provide data protection and security, and should compose with our proposal.

For concreteness, we center our discussion around relational databases, which are central to many web service backends. We first sketch the system design, and then explain how it facilitates key GDPR rights. §3 will discuss how to extend our design to include other systems, such as blob stores and model serving infrastructure.

2.1 System design

Our key idea is for each user to have her own, structured shard of the storage backend (Figure 1). This **user shard** stores all information about this user, such as profile information, posts, uploaded pictures, records of votes or "likes", or other application-specific information. Storing data in a user shard represents an association of control, or, for many data items, ownership. Therefore, a user shard never contains information related to other users, or derived information that combines multiple users' data. We expect that the data subject owning the user shard is often the primary contributor to it, although other entities—including data controllers—may also add data to the shard (consider, *e.g.*, doctors adding to a medical record).

To combine users' data, as most applications require, the backend builds **materialized views** over the user shards. These materialized views are what applications query, and different applications may define different views that suit their needs. For example, a microservice application for the personalized front page of a social application may define a view for the social graph, views that hold users' most recent and most-liked posts, and a view for posts with associated images.

We envision a highly dynamic user shard set. Users will continuously update their shards as they interact with the web service; new users will add shards, while other users remove their shards or withdraw parts of their data. Moreover, we would like a system based on dynamic user shards to enjoy the same performance as today's applications do with an optimized schema. Such optimized schemas often combine multiple users' data in tables that make sense for application semantics, such as a table containing all posts. This requires a system with support for a large number of materialized views

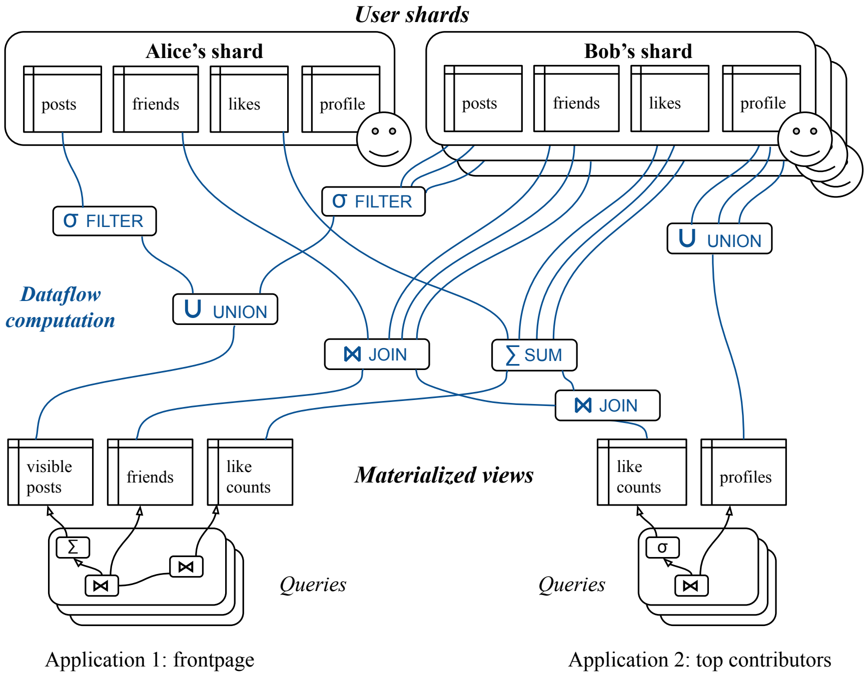

Fig. 1. Example of our architecture for a web service with two applications. Applications write new data to user shards *(top)*. The database processes changes through a dataflow *(middle, blue)* to update tables in the materialized views, which applications *(bottom)* query.

(tens or hundreds of thousands with many users), with efficient dynamic view creation and destruction, and with excellent read and incremental update performance.

We believe that a key enabling technology for making this design efficient already exists. The **partially-stateful dataflow** model supports high-performance, dynamic, partially materialized, and incrementally-updated views over input data [11]. Using this model, we can build a streaming dataflow computation over the user shards, linking each user shard to the materialized views that changes to the shard will affect. Every write to a user shard then becomes a streamed update that flows through this dataflow to update the materialized views; the addition of new user shards becomes a set of batched updates introducing a large collection of new records; and the removal of a user shard becomes a set of batched updates that revoke previously-sent updates. The backend becomes a long-running, streaming dataflow computation with its ground-truth state federated over the user shards. The partially-stateful nature of the dataflow allows the system to proactively update some materialized views (or parts thereof), while reactively computing information in others by querying "backwards" through the dataflow.

Partially-stateful dataflow is a convenient abstraction for materialized views over federated user data for several reasons:

1. the dataflow implicitly represents dependencies between records, such as a post and the likes associated with it via a foreign key;
2. dataflow models (*e.g.*, differential dataflow [17]) allow processing additions, updates, and removals of user shards as incremental computations;
3. dataflow computations can be sharded, parallelized, and scaled with relative ease, making the architecture suitable for scaling to large web services; and
4. partially-stateful dataflow can selectively materialize only parts of downstream views, which keeps the space footprint manageable and allows applications to implement their own caching policies, such as keeping data only for active users, or only for the most popular entities.

The challenges in realizing our design lie in achieving high performance while still providing intuitive consistency semantics for complex applications. Specifically, the user shard structure will yield dataflows over thousands or millions of individual user shards. These dataflows will have extremely "wide" dependencies (*i.e.*, many incoming edges) at points where the computation combines data across users. Query evaluation of partially-stateful dataflow is likely to be slow for such dependencies, since an upwards query through the dataflow must contact many shards. But eager, forward update processing has no such limitation. Semantics of the dataflow execution also matter: in a distributed, streaming dataflow with many servers processing updates in parallel, updates derived from a particular change to a user shard may reach some materialized views before others. An application that writes to a user shard may expect to see its write reflected in subsequent reads, as it would when interacting with a classic database. But providing even this read-your-writes consistency over a large-scale dataflow will result in expensive and unscalable coordination if done naively. Finally, if the dataflow combined data from many user shards, several users may share ownership of derived records in the materialized views. The withdrawal of a user shard may affect these derived results, and the semantics of handling revocation of records that impacted such co-owned derived results need to be clear.

We believe that if it works, our dataflow design will yield a framework for building complex applications while granting users unprecedented control over their data.

2.2 Right to access

Supplying a user with a copy of all her data stored and processed in the system, as required by the GDPR's "right to access" for data subjects (Article 15), is straightforward in a compliant-by-construction design like ours. To serve a data subject's access request, the system simply sends the data subject a copy of her user shard. This simplicity contrasts with post-hoc approaches that extract data using complex, manually-crafted queries or custom crawlers that identify data related to a subject for extraction and manual verification [8]. Achieving this simple compliance-by-construction with user shards imposes only two requirements: first, that the schema of the user shard is free of proprietary information—such as, *e.g.*, the data controller's or processor's backend architecture—and second, that the data subject is permitted see all data in her user shard. We therefore believe that compliance-by-construction systems should assume that a user shard and its structure are visible to the data subject in their entirety.

In addition to access to the raw data, the GDPR right of access also requires that the user be provided with information regarding "the purposes of processing" and "the existence of automated decision-making [. . . and] the logic involved" [9, Art. 15]. Compliance with these requirements is trickier to ensure by construction, since processing purpose and decision making happen—at least partly—in the application code. It might be possible, however, to analyze the dataflow below user shards and generate a description of all materialized views and applications that a given user's data can reach and thereby might affect. Such an analysis would provide an automated means of extracting the information required, and might also facilitate compliance with the GDPR's right to objection, which allows data subjects to reject certain types of processing (see §2.5).

2.3 Right to erasure

The GDPR's Article 17 requires that users must be able to request erasure of their data "without undue delay". In a compliant-by-construction design, this involves removing a user shard from the system. Withdrawing a user shard effectively erases all data contained in it, and then remove or transform dependent downstream information in the dataflow and materialized views.

In principle, removing dependent downstream data is easy as long as the dataflow's operators understand revocations as well as insertions. For example, revoking a vote record for a post from a counting operator involves reducing the count by one, processing a revocation through a join produces revocations for all joining keys, etc. By sending a revocation update for all information contained in the withdrawn shard, the system ensures that all derived downstream information is removed. And since all materialized views queried by applications depend on the dataflow, which itself is a fault-tolerant, distributed computation, we can be assured that all derived information will indeed (eventually) be removed.

All common relational operators have complements compatible with this model, but more complex application semantics may require deeper system support for data removal. For example, consider Alice removing her account from a news aggregator website such as HackerNews or Lobste.rs[3]: removing the user shard in question will remove Alice's posts and votes. More insidiously, the revocation also covers information like invitations to the site that Alice issued, moderation decisions she made (if she's a moderator), and personal messages she sent to other users. Some of this information may need to persist—perhaps in anonymized form—even though Alice originally contributed it!

We believe that dataflow will work even in the presence of these complex application semantics, provided the application developer can express a policy for how each dataflow operator or materialized view handles record revocation. Instead of simply inverting the effect of the original record, the operators may *e.g.*, re-attribute, anonymize, or otherwise transform the derived information.[4] However, the invariants of partially-

[3] `https://lobste.rs`

[4] However, general-purpose "undoing" of computation that extends beyond relational operators can be hard [5, 6]. Imagine, for example, a dataflow operator that trains an ML model on Alice's data: it is unclear how to "invert" the training and revoke Alice's information from the trained model. §3 describes ideas for how we might handle this situation.

stateful dataflow require that any materialized result must also be obtainable by executing a query over the base data. The dataflow system may meet this requirement by creating records that support the transformed derived data, and storing these records in a special shard for deleted users.

2.4 Right to data portability

Compliance with the right to data portability (GDPR Article 20) follows from the combination of the rights to access and erasure. To move data her data from one data controller to another, a user can simply retrieve her user shard from the current controller, withdraw it, and then introduce the retrieved shard to the new controller. All derived information at both controllers will update appropriately, assuming that there is a common data description standard for user shard schemas, or that the controllers know how to transform user shards to and from their respective schemas. This is admittedly a big assumption, but we believe that standardized formats or conversion tools will become available once user data is widely available in the "structured, commonly used and machine-readable format" [9, Art. 20] that the GDPR requires.

2.5 Right to objection

The GDPR also grants users the right to object "any time to processing of personal data concerning him or her" [9, Art. 21] for specific purposes (such as marketing), with some exceptions. With our design, exercising this right involves preventing the flow of data from a user shard into subgraphs of the dataflow that apply specific kinds of processing or which lead to materialized views for specific applications.

We believe that adding appropriate "guard" operators to the dataflow can make it feasible to enforce this right. These operators would check whether a user has objected to the use of her data for *e.g.*, marketing, and prevent any data from an objecting user's shard from affecting views used in marketing workflows. We envision that achieving compliance this way requires applications to augment their materialized view specifications with a declarative specification of each view's purpose, or a reason for overriding the right to objection and processing data without consent (as per GDPR Article 6). We believe that such a declarative specification is far simpler, easier to audit, and more likely to be enforced correctly than adding explicit checks for user objection in application code.

3 Challenges and Opportunities

Realizing our vision raises interesting research questions and its success requires overcoming several challenges.

Classes of personal data. The data associated with a data subject can be contributed directly by that subject into her user shard, but may also originate with other entities. For example, data controllers sometimes create data *about* a user: a government agency may create a birth certificate or tax information, a hospital may create and add to a medical record, or a network operators may collect metadata statistics about the user's network use. The GDPR grants that user the rights of a data subject for this content, requiring the system to store such content in the relevant user shard. This will require intuitive interfaces that allow applications to address the correct user shard on each database write, ideally without requiring intrusive application changes.

Even if the data subject contributed content directly, it may be subject to different policies. For example, a user may both browse and contribute articles to a news site. Browsing data is personal, unshared, and typically subject to strong GDPR protections; meanwhile, a contributed article may be subject to a contract giving the site permission to host the article indefinitely. Furthermore, in some cases, such as shared data, a user's withdrawal from a site might require application-specific anonymization rather than outright data removal. The presence of multiple classes of data in the same user shard could complicate some compliance mechanisms; however, user shards are inherently flexible, allowing such designs as *multiple* shards per user, one per data class.

Shared data. Not all data is clearly owned by a single user. Should a private message on Facebook be associated with one user or both? If my friend deletes her copy of the message from Facebook's database, is my side of our conversation removed entirely, or should a possibly-anonymized ghost message persist in my user shard?

Access control. Even though a user shard contains data associated with a particular data subject, this association may not imply unlimited control. For example, although you may be the data subject of tax records indicating what you owe, you certainly cannot change or remove them! This suggests that controls over the management of data in a user shard need to exist: some information will be immutable to the data subject, but mutable by data controllers; other information may need to persist even when the user removes her shard from the system. To realize the right to portability by retrieving a user shard (as per §2.4), we may need a form of trusted transfer between data controllers, or an assurance mechanism for immutable data if the data subject is part of the transfer. Perhaps the controllers could exchange hashes or signatures of the immutable content, and use these to validate the ported user shard after import.

Schemas. User shards will have a well-defined schema, but this schema may differ significantly from the schemas desired by applications, which often perform queries across groups of user data. The dataflow transformations that combine user shards into views convenient for application access may be complicated; their performance may benefit from insights from literature on partitioned in-memory databases [26].

Changing the user shard schema presents challenges and opportunities, as there will be as many user shards as there are users, or more. On the one hand, user shards represent natural boundaries for gradual schema change deployment; but on the other hand, completing a schema change may require migrating millions of logically- and physically-distinct databases.

Consistency. Current partially-stateful dataflow implementations provide only eventual consistency. This may suffice for some applications, or for many parts of some applications, but strong consistency is important for some parts of all applications. We see this as an opportunity to develop high-performance partially-stateful dataflow implementations that support stronger consistency, *e.g.*, through pervasive multiversioning.

Cross-system abstractions. Our exposition in §2 focused on an RDBMS, but web services rely on many backend systems for storage and data processing. If user shards are the unified ground truth storage of all contributed data, the dataflow over them must feed not merely tabular materialized views, but also diverse endpoints like blob stores, MapReduce jobs, ML training and inference, and others. This creates an opportunity to define cross-cutting abstractions for dataflow between backend systems that augment current datacenter system APIs. It also raises challenges: *e.g.*, how do we revoke training data from an already-trained ML model?

We envisage partially-stateful dataflows that feed data from user shards into consumer systems subject to policies over the interfaces. For example, a policy governing MapReduce jobs may require recording which user shards contributed to the job result, and have withdrawal of any such shard trigger re-execution. Systems might also specify a threshold on shard withdrawals below which the derived effects of data revocation are minimal or provably untraceable (*i.e.*, a notion of differential privacy is guaranteed). This might help, *e.g.*, with ML models trained on a data subject's information: if it is impossible to tell whether an inference came from a model trained with this data or from one trained without it, it is safe to avoid retraining when the subject withdraws her data.

Trust model. In an ideal world, an end-user would never need to trust a data controller or data processor. A strong threat model provides hard guarantees, but often yields systems heavily rely on cryptography and have restrict functionality. Alternatively, we might presume that companies follow the law and faithfully implement laws like the GDPR, and that out-of-band enforcement mechanisms (such as fines) take care of exceptions.

In technical terms, this model shifts the focus from making it impossible to violate privacy laws to easing compliance with them. This can result in low-overhead systems that maintain the functionality users demand, but offers no absolute guarantees.

Specifying privacy policies. The GDPR codifies general responsibilities of a data processor, but leaves it to the data processor to provide specifics in a *privacy policy*. Privacy policy languages designed for computers rather than human lawyers could be a fruitful research direction to ease automated policy enforcement. For example, our proposed system would benefit from machine-parseable privacy policies that specify what dataflows to restrict, how to handle data erasure on shard withdrawal, and what views specific applications are permitted to define.

4 Related work

The desire to give users control over their personal data has been the motivation for considerable existing research. This research addresses a wide variety of use cases, adversary models, and presupposes different standards and ideals for user data protection. The GDPR and similar comprehensive privacy legislation, by contrast, for the first time defined concrete standards that real-world companies must comply with.

Researchers have observed that retrofitting compliance with such wide-ranging regulation onto existing systems and processes is challenging: business models rely on combining data across services, modern machine learning (ML) algorithms violate rights to explanation of automated decisions, and pervasive caching and replication complicate data removal [24]. Minimally-invasive changes that make existing system compliant can substantially degrade their performance [23]. We believe that this inability to retrofit compliance motivates new system designs and new abstractions for inevitable interaction between systems.

4.1 Malicious or negligent data controllers and data processors

Some prior work seeks to protect users against a malicious data processor, using sandboxes [12], by relying on decentralized storage with churn to effect self-destruction of data unless refreshed [10], or by using cryptographic constructs for oblivious computation [29, 31] or computation over encrypted data [21, 22]. These systems have seen limited uptake in practice, perhaps due to the high cost—in terms of overhead and restricted functionality—that strong, often cryptographic, guarantees impose.

Information flow control (IFC) can protect against data breaches by statically or dynamically verifying that it is impossible for specific system components or code to access private user data [7, 14, 34]; programming language techniques similarly ensure that applications handle user data in accordance with a privacy policy [20, 32, 33]. To the same end, multiverse databases [16] compute individualized materialized views for each end-user. These approaches help meet the GDPR's data security requirements, but they do not address other aspects, such as users' rights to access, object, erasure, or data portability, which our proposed design addresses by construction.

4.2 User control over data

Riverbed [30] allows end-users to define policies for their data and enforces them over entire web service stacks: using containers, Riverbed forks a complete stack for every new set of policies. This grants users control over their data, but prohibits and sharing across users with disjoint policy sets, which severely restricts functionality. W5 [13], by contrast, proposes to combine user data in a single platform and has users explicitly authorize access by applications running on this platform, relying IFC to enforce isolation. This achieves good performance, but requires laborious effort on users' part, and is incompatible with web services that may wish to avoid exposing their internal application structure.

Perhaps most similar to our dataflow design are the ideas of "standing queries" over distributed data in Amber [2] and Oort [3]. Structured as a publish-subscribe network, the their focus is to allow cloud applications to access data stored with multiple storage providers, and users are responsible for setting permissions on their data. This is akin to making our user shards held globally queryable; our design instead focuses on enforcing GDPR compliance over data stored within a single web service backend.

BStore [4] and DIY hosting [19] suggest to store users' data lives on cloud storage services (such as Amazon S3), which web applications access through a filesystem interface or serverless functions. Solid [15] and Databox [18] go one step further and have users run personal, physical or virtual, servers that host all data and execute all server-side application logic, combining user data on different “pods” or databoxes via well-defined APIs. These interfaces achieve user control over data, but lack support for the long-running, stateful services at the backbone of today’s web services, which rely on computing and caching derived data. Our user shards are instead held on a web service’s servers (*e.g.*, Google’s), allowing for efficient stateful services, but we envisage APIs for the creation and withdrawal of user shards.

4.3 Data revocation

Removing user data from server-side systems, and revoking its effects on derived information, is a challenging problem. Some prior systems, such as Vanish [10], seek to give users the ability to revoke data even if processed, cached, and stored online and on machines beyond their control. In Vanish, data “self-destructs” after some time unless it is actively refreshed, but revocation is all-or-nothing—*i.e.*, it is impossible to revoke only one of many records that impacted a piece of data stored in Vanish—and Vanish relies on cryptography and a peer-to-peer distributed hash table, making it hard to fit into today’s established web service stacks. Undo computing [5, 6], on the other hand, seeks to provide a general-purpose mechanism to undo only *specific* frontend requests and their derived side-effects. The use case is to undo malicious requests that exploited bugs in a web application, and any secondary effects or subsequent data modifications these requests applied, restoring a “clean” web service backend.

In a compliant-by-construction database, we trust the system (and the data processor who runs it) to faithfully execute shard revocation requests, and expect that fines under the GDPR are sufficient to discourage foul play. In our dataflow design, determinism simplifies undoing requests, as the dataflow’s inherent dependency structure and known operator semantics capture much of the information that undo computing (which covers non-determinism) has to extract from logs.

5 Conclusions

In this position paper, we argued that recent privacy legislation such as the GDPR constitutes an exogenous change that necessitates new system designs, much like changing applications or hardware have in the past.

We proposed a new web service backend architecture that puts users in control of their data, and which aims to be GDPR-compliant by construction. Applied to an RDBMS, our design requires changes to classic schema design and query processing, but leaves the application development model unchanged.

While we believe our ideas indicate a promising direction, and efficient and generalizable implementation requires addressing several research challenges that span databases, distributed systems, programming languages, and security. We are excited to work on these challenges ourselves, and we encourage the community to take them up, as there is plenty of work to do.

Acknowledgments

We thank Jon Gjengset and the anonymous reviewers for helpful comments that substantially improved this paper. This work was funded through NSF awards CNS-1704172 and CNS-1704376.

References

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

[2] Tej Chajed, Jon Gjengset, Jelle van den Hooff, M. Frans Kaashoek, James Mickens, Robert Morris, and Nickolai Zeldovich. "Amber: Decoupling User Data from Web Applications." In: *Proceedings of the 15th Workshop on Hot Topics in Operating Systems (HotOS)*. Kartause Ittingen, Switzerland, May 2015.

[3] 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 Computer Science and Artificial Intelligence Laboratory, Dec. 2016. URL: https://dspace.mit.edu/bitstream/handle/1721.1/105802/MIT-CSAIL-TR-2016-015.pdf?sequence=1.

[4] Ramesh Chandra, Priya Gupta, and Nickolai Zeldovich. "Separating Web Applications from User Data Storage with BSTORE." In: *Proceedings of the 2010 USENIX Conference on Web Application Development (WebApps)*. Boston, Massachusetts, USA, 2010, pages 1–1. URL: http://dl.acm.org/citation.cfm?id=1863166.1863167.

[5] Ramesh Chandra, Taesoo Kim, Meelap Shah, Neha Narula, and Nickolai Zeldovich. "Intrusion Recovery for Database-backed Web Applications." In: *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*. Cascais, Portugal, Oct. 2011.

[6] Haogang Chen, Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. "Identifying information disclosure in web applications with retroactive auditing." In: *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Broomfield, Colorado, USA, Oct. 2014.

[7] 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)*. Vancouver, British Columbia, Canada, Oct. 2010. URL: http://adam.chlipala.net/papers/UrFlowOSDI10/.

[8] Peter Cresse. *The GDPR: Where Do You Begin?* CloverDX Blog. Aug. 2017. URL: https://blog.cloverdx.com/gdpr-where-do-you-begin (visited on July 17, 2019).

[9] "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. URL: `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC`.

[10] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. "Vanish: Increasing Data Privacy with Self-destructing Data." In: *Proceedings of the 18th USENIX Security Symposium*. Montreal, Canada, 2009, pages 299–316. URL: `http://dl.acm.org/citation.cfm?id=1855768.1855787`.

[11] Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, et al. "Noria: dynamic, partially-stateful data-flow for high-performance web applications." In: *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Carlsbad, California, USA, Oct. 2018, pages 213–231.

[12] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data." In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*. Savannah, Georgia, USA, 2016, pages 533–549. URL: `http://dl.acm.org/citation.cfm?id=3026877.3026919`.

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

[14] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. "Information Flow Control for Standard OS Abstractions." In: *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. Stevenson, Washington, USA, 2007, pages 321–334. URL: `http://doi.acm.org/10.1145/1294261.1294293`.

[15] Essam Mansour, Andrei Vlad Sambra, Sandro Hawke, et al. "A Demonstration of the Solid Platform for Social Web Applications." In: *Proceedings of the 25th International Conference Companion on World Wide Web (WWW)*. Montréal, Québec, Canada, 2016, pages 223–226. URL: `https://doi.org/10.1145/2872518.2890529`.

[16] Alana Marzoev, Lara Timbó Araújo, Malte Schwarzkopf, et al. "Towards Multiverse Databases." In: *Proceedings of the 17th Workshop on Hot Topics in Operating Systems (HotOS)*. Bertinoro, Italy, 2019, pages 88–95. URL: `http://doi.acm.org/10.1145/3317550.3321425`.

[17] Frank McSherry, Derek G. Murray, Rebecca Isaacs, and Michael Isard. "Differential dataflow." In: *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR)*. Asilomar, California, USA, Jan. 2013.

[18] Richard Mortier, Jianxin Zhao, Jon Crowcroft, et al. "Personal Data Management with the Databox: What's Inside the Box?" In: *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking (CAN)*. Irvine, California, USA, 2016, pages 49–54. URL: `http://doi.acm.org/10.1145/3010079.3010082`.

[19] Shoumik Palkar and Matei Zaharia. "DIY Hosting for Online Privacy." In: *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets)*. Palo

Alto, California, USA, 2017, pages 1–7. URL: http://doi.acm.org/10.1145/3152434.3152459.

[20] Nadia Polikarpova, Jean Yang, Shachar Itzhaky, and Armando Solar-Lezama. "Type-Driven Repair for Information Flow Security." In: *CoRR* abs/1607.03445 (2016). arXiv: 1607.03445. URL: http://arxiv.org/abs/1607.03445.

[21] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. "CryptDB: Protecting Confidentiality with Encrypted Query Processing." In: *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*. Cascais, Portugal, 2011, pages 85–100. URL: http://doi.acm.org/10.1145/2043556.2043566.

[22] 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. URL: http://dl.acm.org/citation.cfm?id=2616448.2616464.

[23] Aashaka Shah, Vinay Banakar, Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. "Analyzing the Impact of GDPR on Storage Systems." In: *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*. July 2019.

[24] Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. "How Design, Architecture, and Operation of Modern Systems Conflict with GDPR." In: *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. July 2019.

[25] Oliver Smith. *The GDPR Racket: Who's Making Money From This $9bn Business Shakedown*. May 2018. URL: https://www.forbes.com/sites/oliversmith/2018/05/02/the-gdpr-racket-whos-making-money-from-this-9bn-business-shakedown/.

[26] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, et al. "C-store: A Column-oriented DBMS." In: *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*. Trondheim, Norway: VLDB Endowment, 2005, pages 553–564. URL: http://dl.acm.org/citation.cfm?id=1083592.1083658.

[27] Mark Sweney. *BA faces £183m fine over passenger data breach*. The Guardian. July 2019. URL: https://www.theguardian.com/business/2019/jul/08/ba-fine-customer-data-breach-british-airways (visited on July 17, 2019).

[28] Mark Sweney. *Marriott to be fined nearly £100m over GDPR breach*. The Guardian. July 2019. URL: https://www.theguardian.com/business/2019/jul/09/marriott-fined-over-gdpr-breach-ico (visited on July 17, 2019).

[29] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. "Conclave: Secure Multi-party Computation on Big Data." In: *Proceedings of the 14th ACM EuroSys Conference on Computer Systems (EuroSys)*. Dresden, Germany, Mar. 2019, 3:1–3:18. URL: http://doi.acm.org/10.1145/3302424.3303982.

[30] 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. URL: `https://www.usenix.org/conference/nsdi19/presentation/wang-frank`.

[31] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. "Splinter: Practical Private Queries on Public Data." In: *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Boston, Massachusetts, USA, 2017, pages 299–313. URL: `https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/wang-frank`.

[32] Jean Yang, Travis Hance, Thomas H. Austin, Armando Solar-Lezama, Cormac Flanagan, and Stephen Chong. "Precise, Dynamic Information Flow for Database-backed Applications." In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. Santa Barbara, California, USA, June 2016, pages 631–647. URL: `http://doi.acm.org/10.1145/2908080.2908098`.

[33] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. "A Language for Automatically Enforcing Privacy Policies." In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. Philadelphia, Pennsylvania, USA, Jan. 2012, pages 85–96. URL: `http://doi.acm.org/10.1145/2103656.2103669`.

[34] Alexander Yip, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. "Improving Application Security with Data Flow Assertions." In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (OSDI)*. Big Sky, Montana, USA, 2009, pages 291–304. URL: `http://doi.acm.org/10.1145/1629575.1629604`.