Sypse: Privacy-first Data Management through Pseudonymization and Partitioning

Amol Deshpande, amol@umd.edu
University of Maryland, College Park, MD, USA

ABSTRACT

Data privacy and ethical/responsible use of personal information are becoming increasingly important, in part because of new regulations like GDPR, CCPA, etc., and in part because of growing public distrust in companies' handling of personal data. However, operationalizing privacy-by-design principles is difficult, especially given that current data management systems are *designed* primarily *to make it easier and efficient to store, process, access, and share vast amounts of data.* In this paper, we present a vision for transparently rearchitecting database systems by combining *pseudonymization*, *synthetic* data, and data *partitioning* to achieve three privacy goals: (1) reduce the impact of breaches by separating detailed personal information from personally identifying information (PII) and scrambling it, (2) make it easy to comply with a deletion request ("right to be forgotten") through overwrites of portions of the data, and (3) reduce the need to access PII for developers or engineers. We present a general architecture as well as several potential strategies for achieving the goals, and some initial experimental results comparing the performance of the different strategies. We end with a discussion of some of the major research challenges moving forward.

ACM Reference Format:
Amol Deshpande, amol@umd.edu. 2020. Sypse: Privacy-first Data Management through Pseudonymization and Partitioning. In *Proceedings of 11th Annual Conference on Innovative Data Systems Research (CIDR'21).* .

1 INTRODUCTION

Data privacy and responsible data stewardship, after being an afterthought in the rush to capitalize on the promise of Big Data, are rapidly growing in importance. Regulations like the European General Data Protection Regulation (GDPR), California Consumer Protection Act (CCPA), Brazil's Lei Geral de Proteção de Dados (LGPD), etc., require organizations to be more transparent about their data collection and usage practices; more regulations along the similar line have either recently passed (e.g., CCPA 2.0) or are in the process of being enacted (cf. [7, 15, 25, 26] for a more detailed exposition of the regulations). At the same time, increased public awareness has led many companies to take a closer look at their data practices to avoid long-term reputational harms. However, a major impediment to implementing better data practices is that, current data management systems and ecosystems are *fundamentally designed to make it easy and efficient to collect, process, and analyze vast amounts of data in a centralized fashion [27].* Although there has been much work in recent years on techniques like differential privacy and encrypted databases, those have not found wide adoption and further, their primary target is untrusted environments or users. We believe that operational databases and data warehouses in trusted environments, which are much more prevalent, also need to be redesigned and re-architected from the ground up to embed *privacy-by-design* principles and to enforce better data hygiene. It is also necessary that any such redesign not cause a major disruption to the day-to-day operations of software engineers or analysts, otherwise it is unlikely to be widely adopted.

In this paper, we present such a redesign of a standard relational database system, called *Sypse*, that uses "pseudonymization", data partitioning, and synthetic data, to achieve several key privacy goals without undue burden on the users.

Pseudonymization refers to de-identifying data by stripping personally identifying fields like *name*, so that the resulting data cannot be tied to a specific individual without additional information. Unlike properly "anonymized" data, pseudonymized data is typically still considered "personal" data[1] since it is usually possible to re-identify the individuals through use of auxiliary data [6, 19]. However, pseudonymization is widely seen as a practical way to reduce the impact of data breaches, and to make it easier to share data internally [5, 11, 20][2]. Pseudonymization also offers a practical solution to the challenges in deleting information in response to an individual's request "to be forgotten". For many practical and business reasons, it is usually not possible to actually delete every single piece of information associated with the individual. A preferred approach instead is **to overwrite a portion (or all) of the data for the individual**, so that it is difficult or impossible to re-associate the information to that individual. The amount of information that needs to be overwritten depends on the type of the data, business requirements, and legal interpretations, but it is usually a small fraction of all the data for the individual, typically the direct identifiers. This approach is not only easier to implement, but also maintains much of the utility of the data for analytics and other purposes like model training. Pseudonymization can dramatically reduce the amount of data that needs to be overwritten, especially if used liberally to not only de-identify the data but also break connections between data items as we will discuss later.

The second core idea underlying Sypse is **data partitioning to minimize impact of data breaches and to simplify processing of the deletion requests**. Monolithic centralized databases are often desirable and are a norm today because of ease of management and querying. Such databases are typically backed up as

[1]This does depend on the regulation; GDPR typically treats such data as personal data, whereas CCPA, HIPAA, etc., appear to exempt de-identified data from the regulation.
[2]"The application of pseudonymisation to personal data can reduce the risks to the data subjects concerned and help controllers and processors to meet their data-protection obligations"; EU GDPR Recital 28.

a unit, and may be duplicated as a unit for debugging, resolving support issues, or performance analysis. Even with very good data practices, this significantly increases the impact of a data breach (which are increasing in frequency despite cybersecurity advances). Backups also pose a significant challenge to handling an individual's request to delete their personal data; it is near impossible to identify and scrub the personal data of the individual from all copies of the database.

Finally, unlike other approaches (e.g., encryption) where the utility of the data may be low to non-existent in absence of the keys, Sypse uses carefully generated **synthetic** data so that the database can still be useful even if some data is pseudonymized or hidden. In many scenarios, the available information may be sufficient to perform debugging or support or performance analysis tasks, but without such synthetic data, applications running on top of the databases are unlikely to continue to function properly.

Sypse is designed as a **stateless layer that sits atop traditional databases** and transparently partitions the data into two or more partitions so that personally identifying information is separated from the detailed personal information. In addition, the primary and foreign keys are also scrambled to achieve additional protection against re-identification. We envision that Sypse will continuously analyze the schema and the data to automatically make decisions about how to partition the data, while allowing the users to override those decisions at any point. This will allow Sypse to provide protections by default without any input from the users. Sypse naturally supports multiple levels of access, and as discussed above, has a design goal that queries should always succeed. All of these design goals are intended to reduce the friction in deploying Sypse; in fact, we hope that Sypse can serve as a drop-in replacement for an existing database. However, there are many research challenges that need to be successfully addressed to realize that vision; in particular, there are significant performance implications of the data scrambling and partitioning that need to be mitigated.

Outline: We begin with describing the Sypse architecture in more depth and motivate the key design decisions we made. We then sketch a variety of partitioning strategies that all achieve the stated privacy goals but have very different performance characteristics. We present a preliminary experimental validation of the key ideas, and end with a discussion of research challenges moving forward.

2 RELATED WORK

There has been much work on related topics like privacy-preserving querying and data mining, statistical notions of privacy (k-anonymity, l-diversity, differential privacy, etc.), secure or encrypted databases, secure multi-party computation, and verifiable databases. The approach we take here addresses a different set of concerns, and can be seen as complementary to most of that work.

Differential privacy enables statistical analysis of data without revealing information about specific individuals, and there has been an intense amount of work on using that core concept for privacy-preserving machine learning, generation of synthetic data, and so on [8, 13, 23]. There is also recent work on expanding the types of SQL queries that can be handled through differential privacy [12, 14]. However, differential privacy, by itself, doesn't directly support any of the goals we outlined above, since the original database with

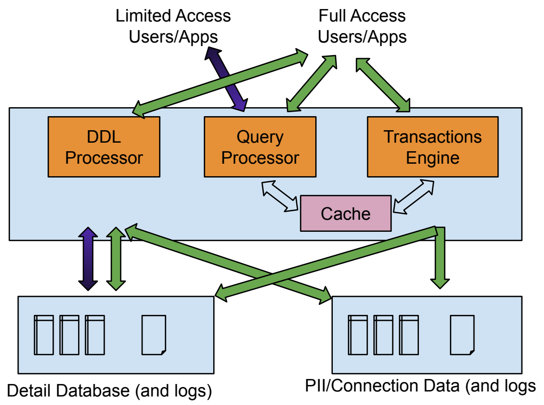

Limited Access Users/Apps
Full Access Users/Apps
DDL Processor
Query Processor
Transactions Engine
Cache
Detail Database (and logs)
PII/Connection Data (and logs

Figure 1: High-level Sypse Architecture w/ 2 Partitions

the personal information is still available to the trusted users and applications. However, we can significantly increase the utility of the synthetic data used in our approach through a more systematic differential privacy-based approach.

Secure/encrypted databases target a scenario where the database is hosted by an untrusted server (e.g., in the cloud) and we wish to support queries over that data while minimizing information revealed to the untrusted server. Broadly, this can be done through the use of encryption techniques that support computations (e.g., Homomorphic Encryption) [3, 10, 18, 22], or through use of trusted hardware like Intel SGX [2, 24]. There is also increasing work on using secure multi-party communication to support federated data analysis across organizations without sharing the actual data [4, 9, 29], and on verifiable computations [1, 30]. Our approach is designed for a trusted environment, and our goal is to avoid inadvertent leakage of data and make it easy to support data and access minimization. The ideas we explore here, including partitioning of data and use of pseudonymization and synthetic data, can be combined with those approaches to provide stricter guarantees.

In recent years, there have been several efforts aimed at redesigning data management systems, analogous to our work here, in light of GDPR and other privacy regulations. Schwarzkopf et al., [25] propose creating per-user data shards for fine-granularity access and compliance management. Kraska et al., [15] explore several approaches to handle requests to be forgotten and other challenges in supporting GDPR; they also propose the use of surrogate keys to support lazy deletion of personal data (our approach goes further beyond by, in effect, adding surrogate keys throughout the database). GDPRBench [26] proposes a new benchmark for evaluating databases against GDPR requirements. ScrambleDB [16] also makes heavy use of pseudonymization, however, it targets a distributed use case and doesn't provide solutions to the three problems mentioned above. Sieve [21] presents approaches to enable fine-grained access control in support of privacy regulations. Most of these projects are currently in preliminary stages making it hard to do a direct comparison, but to our knowledge, none of the prior work has undertaken a systematic evaluation of data partitioning and pseudonymization to achieve the privacy goals.

3 ARCHITECTURE

The core idea underlying Sypse is to **partition the data into two or more partitions, while introducing pseudonymous identifiers and synthetic data to allow differentiated access to the data**. Figure 1 shows the high-level architecture of Sypse as a layer on top of a standard relational database system, where the data is split into two partitions, and two access levels are supported. For either access level, the same *schema* is presented to the user, and any SQL queries can be executed against that schema. However, for the limited access level, the queries will be run against *pseudonymized* or *synthetic* data only (we discuss data utility later in the section). Updates are currently only permitted under full access; however, in theory, updates to the non-PII data could be supported under limited access. Note that, in the general case, we may have more partitions and more access levels, and those two, in general, are independent of each other (e.g., you may have more than 2 access levels with just two partitions).

In the rest of the paper, we assume that the data is split into two databases as shown in the figure, that we will term **Detail Data(base)** and **PII Data(base)** for simplicity.

The data partitioning obeys the following design principles:

- Any personally identifying information (PII) and any connection information that is needed to group/aggregate detail data and/or connect detail data to PII is concentrated in the PII Database. We discuss what constitutes PII in more detail below.
- All information that can be treated as non-PII information is maintained in the Detail database. We re-emphasize that **this information still needs to be treated carefully since it is only "pseudonymized" and not "anonymized"**. In particular, this database **does not satisfy any of the statistical privacy requirements like differential privacy**, and thus the access controls on it cannot be relaxed.

The key components of Sypse are:

(a) **DDL Engine**, responsible for intercepting any schema update statements, and appropriately creating the tables in the underlying databases;
(b) **Query Processing Engine,** that executes the user queries by using the tables from the underlying databases as appropriate for the access level; and
(c) **Transaction Engine**, for handling the updates to the database by appropriately routing the updates to the right tables and generating synthetic or pseudonymized data as required.

As we will discuss later, for performance reasons, the query processing and transaction engines will likely need to use in-memory caches to avoid expensive joins; such caches would typically be built during initialization. Furthermore, a key requirement of Sypse is that the **underlying databases are administratively separated, have different access controls, and are backed up independently**, to reduce the possibility of multiple of those being breached simultaneously. This may be achieved, e.g., through geographic separation or use of virtual private clouds. This adds further performance considerations because of increased distributed transactions and higher query execution overheads; however, all of these are issues that the database community has developed techniques to mitigate over the years.

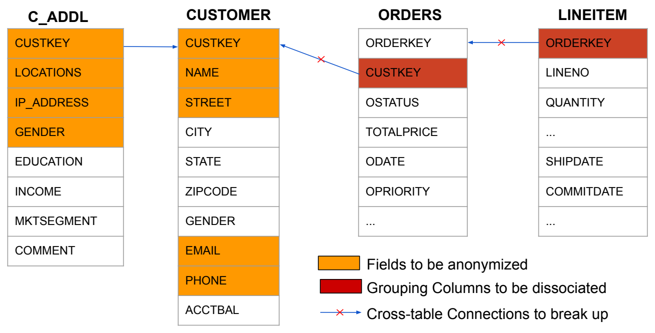


Figure 2: Example Annotated Schema

3.1 Personal Data vs PII vs De-identified Data

Before diving deeper into discussing how data should be split across the two databases, it is necessary to better understand what we mean by personal data (PI) vs personally identifying data (PII), and what kinds of information we need to pseudonymize or hide. We use a familiar TPC-H-like example (Figure 2) for this purpose, augmented with an additional table.

- First, data fields like *name*, *address*, *custkey*, *IP Address*, etc., can be directly tied to a specific individual (or to a small subset), especially given the abundance of auxiliary information easily available now-a-days (see NIST Guidance for a more detailed discussion of PII [17]). Such fields should be moved to the PII Database and replaced with pseudonymous identifiers (also called *surrogate keys* [15]) to separate them from the other data fields. Any groups of data fields that together are sufficiently unique should also be treated similarly.
- Geolocation or biometric information also tends to be easy to re-identify and should be treated similarly.
- **Groups information:** Even if *customer* PII is pseudonymized, analyzing the collection of *orders* made by a single *customer* (i.e., all *orders* with the same *custkey* together) leads to a loss of privacy, including possible re-identification. Hence, we may wish to break up these groups so that multiple orders cannot be tied to the same individual using Detail Database alone. We note that presence of *near-keys* (i.e., approximate functional dependencies) can also lead to a loss of privacy and those may need to be explicitly handled as well.
- Finally, depending on the level of pseudonymization desired, we may wish to explicitly break the *connections* between tables (this may be unnecessary if groups are already broken up). For example, we may wish to break the connection between the two *customer* tables in the example to remove the possibility of combining the fields across those to re-identify individuals. How far this should be done depends on the sensitivity of the data – intuitively speaking, the "farther" a connection is from the PII data, less the need for it to be broken up.

In our initial prototype, we assume that we are provided with an annotated schema where the data fields, foreign key references, and groups are designated with the desired guarantees, so that we can focus on the implementation and validation of the approach. We envision that this will be done largely automatically and transparently – identifying sensitive and PII data fields is relatively straightforward, but group/connection information requires more analysis because of a *lack of schema discipline in practice*.

We re-emphasize that the goal here is not to achieve true anonymity, but rather to enable flexibly adding pseudonymization so that the potential for inadvertent misuse is reduced, and do this while reducing the friction typically encountered in operationalizing such approaches.

We begin with a brief discussion of the privacy guarantees and trade-offs and then present three strategies in the next section.

3.2 Privacy Guarantees

The privacy guarantees provided by Sypse depend on the specific decisions that are made regarding the columns and connections to be hidden. Overall the guarantees will be significantly weaker than differential privacy, but on the other hand, Sypse imposes no constraints on the types or numbers of queries that can be performed.

More specifically, for someone with access to both the databases, there is clearly no privacy protection (beyond what might be implemented in the database itself). If someone only has access to the Detail database, they cannot directly learn any of the PII information or correlate information across tables or within a table as per the decisions made in the annotated schema. However, since much of the non-PII data is still visible in the Detail database, there is potential for combining that with auxiliary information to re-identify individuals (e.g., if one knew that an individual visited a specific store on a specific date, the orders by that consumer may be identifiable). On the other hand, if someone only has access to the PII Database, much of the detailed information is hidden from them, but there is obviously a significant loss of privacy.

3.3 Trade-offs

There are many ways to design **partitioning strategies** that can support the privacy goals above; one of our goals with this project is to evaluate the trade-offs in a systematic manner.

First, different strategies may have significantly different characteristics with respect to **query** and **transaction performance**. Joins involving real data, where data across the two databases must be combined, are especially problematic – the desire to keep the PII Database small naturally makes it difficult to evaluate joins efficiently. Similarly, a single transaction may have to involve two or more different databases. Even if the databases are co-located on the same machine, there may be process or context boundaries that need to be spanned. Clearly a fraction of the transaction workload is localized to one or the other database (e.g., *address* updates only affect the PII database, whereas updating *parts* only affects the Detail database). However, other transactions (e.g., adding a new *order*) may require updates to both databases. Ideally we'd like to minimize the fraction of the workfload where this is needed.

Second, the **utility of the detail dataset** by itself is a key concern that will impact adoption. A large class of queries or analysis tasks can potentially be performed successfully without access to the PII information. This includes statistical analysis of the distributions of non-PII data fields, many machine learning and data mining tasks, support or debugging tasks (against production data), and possibly performance analysis. However, the design decisions made during the partitioning phase will dictate what fraction of the workload can be supported on the Detail Database alone.

Third, the overall increase in the storage footprint as well as the **size of PII database** are key considerations. Since the PII Database is more sensitive, we would like its size to be as small as possible, so that access control and backups are more manageable. For example, it may be acceptable for the Detail database backups to be offloaded to tape storage, but ideally the PII database backups are not stored on tape because of the difficulties of deleting data there. More stringent access controls, richer auditing, and even review processes to approve access requests, may be used for the PII Database.

Finally, an important consideration is how difficult and expensive it is to **forget an individual**. We assume that it is sufficient to delete the information in the PII Database to forget an individual; as we discussed earlier, this is widely considered to be sufficient in practice. This means that the pseudonymized data associated with that individual in the Detail database, will still be maintained; thus the partitioning strategy needs to be designed and configured appropriately to ensure that the remaining, pseudonymized data is difficult or impossible to trace back to the individual. We note that our approach provides the users with full control on what data is kept in the PII database, and how much the rest of the data is pseudonymized. In theory, every single data atom about a person could be dissociated from other data atoms to achieve extreme pseudonymization.

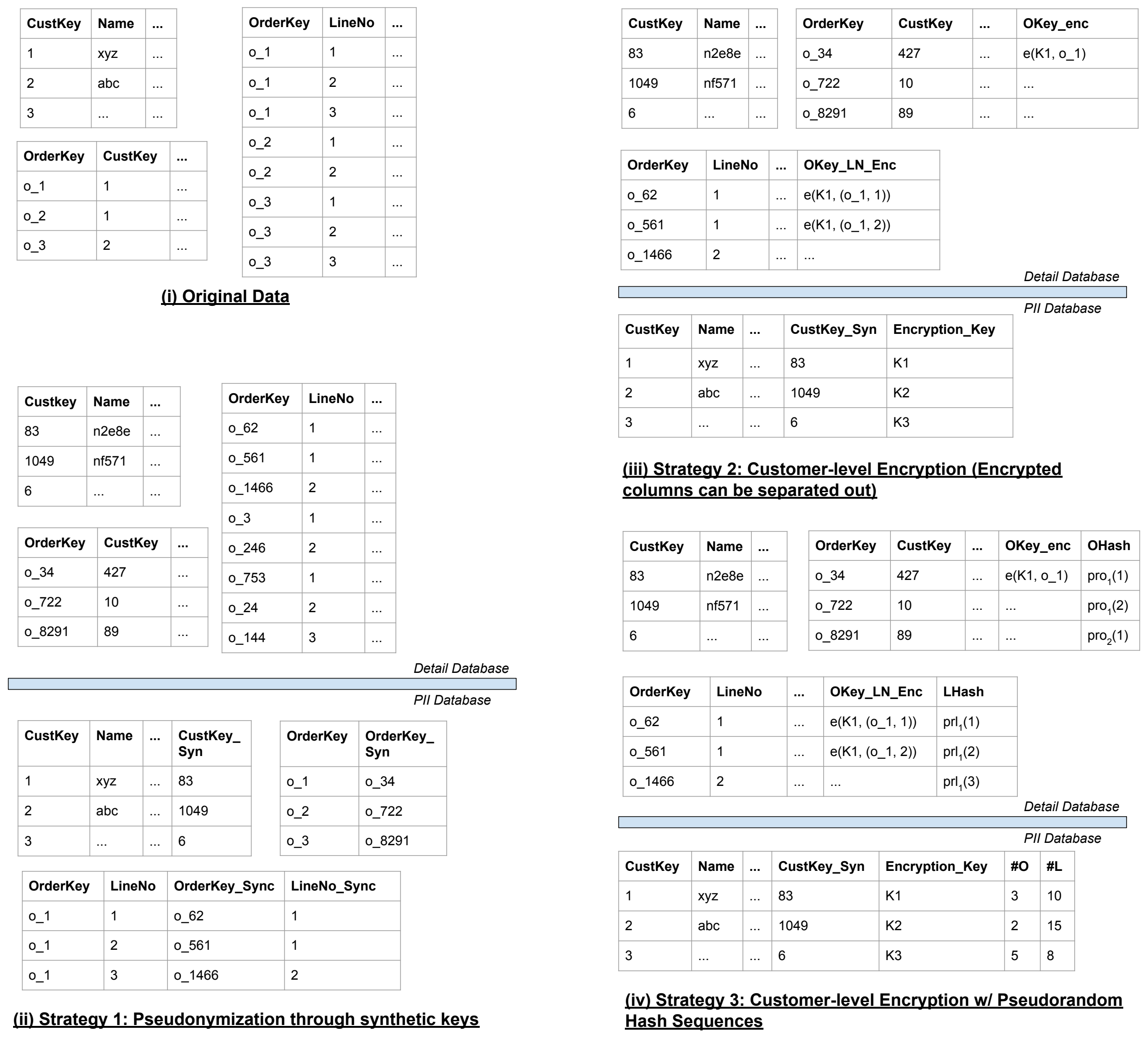
4 PARTITIONING STRATEGIES

In this section, we sketch a few different strategies that can be used to achieve the goals above, and discuss their pros and cons with respect to the desiderata above.

4.1 Strategy 1: Duplicated Tables and Columns

The simplest strategy that satisfies the goals above is to copy out a subset of the columns into the PII database, and replace them with synthetic data. This includes any columns that contain PII or sensitive data, and any foreign key and grouping columns designated for pseudonymization. Figure 3(ii) shows this for a simple three-table schema. Specifically:

- Any table containing PII is created in both databases – the table in the Detail database contains all the columns but with the PII fields replaced with synthetic data, whereas the PII database table contains only the PII columns along with the primary key. If the primary key is marked as PII, then the PII database table also contains synthetically generated keys (as shown in the example). In our initial prototype, we generate the synthetic data randomly, but ideally it matches the real data types and distribution as closely as possible to increase the utility of the Detail database by itself.
- If an FK association is to be broken up (e.g., *order* to *customer*), then the corresponding FK column is modified in the Detail database to point to a random tuple in the referenced relation. If we still wish to maintain the grouping of *orders* by *custkey*, then this pseudonymization is done consistently (i.e., a given *custkey* is replaced with a randomly generated



(i) Original Data

CustKey	Name	...
1	xyz	...
2	abc	...
3	...	...

OrderKey	CustKey	...
o_1	1	...
o_2	1	...
o_3	2	...

OrderKey	LineNo	...
o_1	1	...
o_1	2	...
o_1	3	...
o_2	1	...
o_2	2	...
o_3	1	...
o_3	2	...
o_3	3	...

(ii) Strategy 1: Pseudonymization through synthetic keys

Detail Database

Custkey	Name	...
83	n2e8e	...
1049	nf571	...
6	...	...

OrderKey	CustKey	...
o_34	427	...
o_722	10	...
o_8291	89	...

OrderKey	LineNo	...
o_62	1	...
o_561	1	...
o_1466	2	...
o_3	1	...
o_246	2	...
o_753	1	...
o_24	2	...
o_144	3	...

PII Database

CustKey	Name	...	CustKey_ Syn
1	xyz	...	83
2	abc	...	1049
3	...	...	6

OrderKey	OrderKey_ Syn
o_1	o_34
o_2	o_722
o_3	o_8291

OrderKey	LineNo	OrderKey_Sync	LineNo_Sync
o_1	1	o_62	1
o_1	2	o_561	1
o_1	3	o_1466	2

(iii) Strategy 2: Customer-level Encryption (Encrypted columns can be separated out)

Detail Database

CustKey	Name	...
83	n2e8e	...
1049	nf571	...
6	...	...

OrderKey	CustKey	...	OKey_enc
o_34	427	...	e(K1, o_1)
o_722	10	...	...
o_8291	89	...	...

OrderKey	LineNo	...	OKey_LN_Enc
o_62	1	...	e(K1, (o_1, 1))
o_561	1	...	e(K1, (o_1, 2))
o_1466	2	...	...

PII Database

CustKey	Name	...	CustKey_Syn	Encryption_Key
1	xyz	...	83	K1
2	abc	...	1049	K2
3	...	...	6	K3

(iv) Strategy 3: Customer-level Encryption w/ Pseudorandom Hash Sequences

Detail Database

CustKey	Name	...
83	n2e8e	...
1049	nf571	...
6	...	...

OrderKey	CustKey	...	OKey_enc	OHash
o_34	427	...	e(K1, o_1)	$pro_1(1)$
o_722	10	...	...	$pro_1(2)$
o_8291	89	...	...	$pro_2(1)$

OrderKey	LineNo	...	OKey_LN_Enc	LHash
o_62	1	...	e(K1, (o_1, 1))	$prl_1(1)$
o_561	1	...	e(K1, (o_1, 2))	$prl_1(2)$
o_1466	2	...	...	$prl_1(3)$

PII Database

CustKey	Name	...	CustKey_Syn	Encryption_Key	#O	#L
1	xyz	...	83	K1	3	10
2	abc	...	1049	K2	2	15
3	...	...	6	K3	5	8

Figure 3: Example showing the three strategies and the tables created; $pro_1(1)$ refers to the first value in the pseudorandom sequence pro with 1 as the seed.

custkey throughout *orders*). A two-column table will need to be added to the PII Database to maintain the real mapping.

- Similarly, if a grouping association is to be scrambled, then the corresponding columns are copied out into the PII database, and replaced with synthetic data in the Detail database. This may entail adding additional rows to the tables depending on the other choices being made. For instance, in the example shown above, say we wished to break the grouping of *orders* based on *custkey*, but we do not wish to break the association between *orders* and *customers*. In that case, every *customer* tuple will need to be duplicated as many times as the number of *orders* associated with it.

We note that the Detail database contains approximate functional dependencies and temporal correlations (e.g., different *lineitems* for a given *order* have ship dates that are temporally close to each other). Most of these issues can be addressed with additional pseudonymization and more aggressive use of synthetic data (and possibly duplication of data along the lines of *oblivious RAM*), at the expense of decreased utility. We envision a user interface that surfaces such potential leakage avenues to the users so they can make practical decisions about what they are comfortable with. As observed by many researchers, there is "no free lunch in data privacy" [13].

Analysis: This strategy is relatively simple to implement and use.

- (+) Generally speaking, querying and analyzing real data is relatively efficient here. In most cases, additional joins are required to "reconnect" the tuples, but the mapping tables are small (2-column in many cases), and we can draw upon the techniques developed in the query optimization literature to optimize those. As an example, a simple aggregate query that requires joining three tables *customers*, *orders*, and *lineitem*, will now require joining 5 of the tables shown in Figure 3(ii) (all but the *customer* table in Detail database). Some of these joins can be avoided by adding extra columns to the PII Database tables – we plan to explore this in more detail in future work.
- (-) Most transactions require updates to both databases. For example, adding a new *order* requires updating both *orders* tables (to add in the corresponding mappings).
- (-) The PII Database size increases proportionally to the Detail database, because of the mapping tables. This negatively impacts one of the primary performance goals of reducing the PII Database size to simplify tracking of backups.
- (-) Forgetting a *customer* requires removing information from the *customer* table in the PII Database, and also modifying the other two PII Database tables so that the grouping information is removed, i.e., so that we can not correlate different *orders* or *lineitems* for the same customer.

Variations: We note that there is a large search space of variations that achieve the same level of pseudonymization with different performance trade-offs (e.g., some of the joins can be avoided by adding more columns to the PII Database tables). Understanding this search space better, and picking the best option for a given annotated schema is a rich area for further research.

Strategy 2: Encrypted Columns: Here, we encrypt all the PII and connecting columns and add these to the same tables, and maintain the encryption keys in the PII Database. One extreme here is to use a single encryption key for the entire database; however, in addition to the issues discussed below, it also has the disadvantage that forgetting a person is difficult and requires updates throughout the database and backups; further, compromise of the single encryption key leaks the entire database. Instead, we evaluate an option where per customer encryption keys are used, and maintained in the PII database indexed by the customer key.

Specifically, most of the PII data fields, FKs, and grouping columns are encrypted with customer-specific encryption keys generated when the *customer* tuples are inserted. We use symmetric encryption in our prototype implementation. The PII Database only contains a lookup table to find the encryption key for a given *custkey*, and the PII fields from the *customer* table (although the *customer* PII fields can be similarly encrypted and maintained in the Detail database itself, maintaining them in the PII database is more efficient and doesn't significantly increase the size of the PII Database, which remains proportional to the size of the *customer* table).

Analysis: Although relatively simple to implement and with a rich history of prior work on related topics [22], this approach has several practical issues with respect to the privacy goals we aim to satisfy.

- (-) Joins involving real data are fundamentally very difficult – for a given encrypted *order* or *lineitem*, we need to iterate through all the encryption keys to identify the correct one (joins over synthetic data can be done as normal). Any additional hints provided to simplify this search in the Detail database reduce the pseudonymization guarantees. For instance, if we were to maintain a pointer to the right encryption key for an encrypted value in the Detail Database itself, that can be used to associate different data items for the same customer. On the other hand, if we were to keep these pointers in the PII Database, the approach doesn't provide any real advantages over Strategy 1. Although there has been much work on joins over encrypted data, that work is in an untrusted setting and does not (to our knowledge) work with per-customer keys.
- (+) Forgetting a customer is very easy – we just need to delete the encryption key and the rest of the PII fields corresponding to the customer.
- (+) The size of the PII Database is also much smaller in this case (proportional to the number of *customer*s), making it easier to manage and track the backups.

Variations: There are a number of variations of this basic approach, that differ in where and how the encryption is performed, and what data is maintained in the PII Database to address the performance issues. We plan to explore those issues in more depth in future work, especially in settings where multiple levels of access are desired.

Strategy 3: Pseudorandom Sequences: This approach attempts to strike a balance between the above two by still keeping the PII Database small, but making joins more efficient. We do it by using per-customer **deterministic pseudorandom sequences** (that can be regenerated on demand) to make it much more efficient to find the encryption key for a given encrypted data item. Specifically, as in Strategy 2, the Detail database is similarly encrypted, with synthetic columns added in to allow join queries to return results w/o access to the PII Database. All the PII and foreign key fields that need to be hidden are encrypted with a customer-specific encryption key that is maintained in the PII Database. However, unlike Strategy 2, with each *order* tuple, we also add a new column that contains a value from a pseudorandom sequence deterministically generated with *custkey* as the seed. Similarly, with each *lineitem* tuple, we add a new column using a separate pseudorandom sequence generated with *custkey* as the seed. Then, in the PII Database, along with maintaining the encryption key for each *customer*, we also keep track of the length of the two pseudorandom sequences.

The main benefit of using a pseudorandom sequence is that, without the seed, it is impossible (cryptographically speaking) to correlate, say, two different *orders* for the same *customer*. But, as we discuss below, given the seed, joins involving real data become much more efficient.

Analysis: This approach is more complex to implement than either of the above, but as we discuss in our preliminary evaluation, it offers a nice balance of the trade-offs.

- (+) Joins involving real data are significantly faster here. At runtime, the information maintained in the PII Database is used to create the relevant pseudorandom sequences, which can then be "joined" with the Detail Database tables to match the encryption keys with the encrypted data. An added benefit is that some joins can be skipped. For instance, for a query that joins the three tables but only projects on, say *c_name* and *l_shipdate*, there is no need to consult the orders table at all, which is only needed to connect the customers with their lineitems.

- (+) Forgetting a customer simply requires deleting the corresponding record from the PII Database.
- (+) The PII Database is somewhat larger than Strategy 2, but still much smaller than Strategy 1.
- (-) Inserts involve more steps since the pseudorandom numbers also need to be generated.

5 PRELIMINARY EXPERIMENTS

We present some very preliminary results from a prototype that we are building on top of PostgreSQL. The goal of this prototype is to validate the key ideas and better understand the different tradeoffs. We attempt to push as much of the computation as possible to the underlying database system. Although this is easy to do for insert/delete/update operations, for any queries across the two databases, much of the computation is done in memory within the wrapper. As we see, this has significant performance implications. Our goal is to work towards an implementation entirely within PostgreSQL after the key ideas are validated.

We use the small subset of TPC-H relations (SF = 0.1) used throughout this paper, i.e., *Customer*, *Orders*, and *Lineitem*. We keep the entire set of attributes from TPC-H except for FKs that refer to other relations. For strategy 2, we use the Fernet Symmetric Encryption package to encrypt the columns with randomly generated per-customer keys. For strategy 3, we also use the MD-5 hash function to generate the per-customer pseudorandom sequences.

We report results for 4 operations, all over real data: (a) Insert a new *customer*; (b) Insert a new *order*; (c) Forget a *customer*; and (d) a query that joins the three relations and projects the result on *l_shipdate*, *c_name* (as noted earlier, for all of the strategies, any queries run under limited access are executed just against the Detail Database directly, and do not suffer any performance penalty). For the microbenchmarks, we report the median across 100 or 1000 operations, and for the query, we report the median across 10 runs. All the experiments were done on a quad-core laptop.

Table 1 shows the results for four operations as well as the database sizes, for the three strategies above and the baseline of not using any of these strategies. We note that the numbers of the three strategies have minimal optimizations; in particular, no caches are being used, and any outside database operations are performed in Python, an interpreted language. Especially for inserts, bulk of the time is in choosing unused surrogate keys so that referential integrity constraints are not violated.

As we can see, the PII Database size for any of the strategies is significantly lower, and this benefit would be much higher if there were more than 3 tables in the database. For the TPC-H Schema, many of the remaining tables like *part*, *supplier*, etc., don't contain personal data and would not contribute anything to the PII Database, while increasing the overall Database size. As expected, PII Database size is the smallest for Strategy 2 and Strategy 3 since

	Baseline	**Strategy 1**	**Strategy 2**	**Strategy3**
Total Database Size	**163 MB**	212 MB	309 MB	260 MB
PII Database Size	163 MB	42.7 MB	**5.7 MB**	**6.1 MB**
Insert Customer	**0.359 ms**	10 ms	**0.733 ms**	9 ms
Insert Order	**0.394 ms**	98 ms	120 ms	116 ms
Forget Customer	181 ms	179 ms	**0.37 ms**	**0.39 ms**
Select Query	0.862 s	7.84 s	est. 34 hours	2.822 s

Table 1: Preliminary Results

we only need to maintain the per-customer keys in that partition (and the lengths of pseudorandom sequences for Strategy 3). The slightly larger overall Database size for Strategy 2 over Strategy 3 is because one column in *orders* and *lineitem* tables (corresponding to real customer key) is stored as a encrypted value in Strategy 2 and as a hash value in Strategy 3 (the former, in our implementation, requires more bytes).

The performance of insert and delete operations matches our expectations – in particular, deletes are much faster for Strategies 2 and 3 compared to the Baseline or Strategy 1. Since personal data footprint typically spans more than 3 tables, the performance gap would be proportionally higher as the number of tables increases; for both Strategies 2 and 3, the cost would remain flat, but would go up significantly for the other two strategies.

Finally, the numbers for the select query show the extreme penalty for Strategy 2 which requires cross-products. Strategy 3 surprisingly outperforms Strategy 1 here – this is because Strategy 1 requires a large number of joins, whereas Strategy 3 only requires one join and wins out despite the overhead of reconstructing the pseudorandom sequences.

6 RESEARCH CHALLENGES

There are many research challenges in building privacy-first database systems like Sypse. We enumerate a few specific research challenges that we are currently investigating.

Automated Schema Analysis: For Sypse to be widely adopted in practice, many of the steps need to be automated and fully transparent to the users of the system. We envision that the decisions about which fields to pseudonymize, which connections to break, etc., are all made automatically through a continuous analysis of the database schema and the data within it (with the ability for the users to see and alter those decisions at any time). That way, a baseline privacy protection is always offered as a best practice, whether or not the users are aware of it. Given the *lack of schema discipline in practice* [28], this is challenging to do. In particular, decisions about personally identifying information, foreign keys (which are rarely noted down), and approximate functional dependencies may require analyzing the actual data and reconstructing the underlying entity-relationship structure. There is significant literature on these topics that we can build upon.

Multiple Personal Entities: A database typically contains multiple distinct personal entities. For example, a TPC-H-like database may contain a separate *employees* relation; personal data associated with those will need to be pseudonymized independently. Understanding the different types of entities in a database, and how they relate to each other is a rich area for further work (e.g., what happens to a data item that is associated with one customer and one employee if the customer data is to be forgotten, is unclear).

Systematic Evaluation of > 2 **Partitions:** Increased number of partitions of the data will lead to better protection against breaches or unauthorized accesses. It is however a significant challenge to generalize the core approach above to support a larger number of partitions in a flexible, data-dependent manner. Even two partitions leads to, as we discussed, a large number of partitioning strategies. Understanding and narrowing the search space to design effective strategies for more than 2 databases is a major research challenge.

Performance Issues: Our preliminary results, while confirming that the approach is feasible, also point out many performance issues. We need to develop techniques for more efficient generation of synthetic data, better join algorithms and query planning techniques, more efficient multi-site transactions, and also caching strategies that don't compromise the privacy guarantees.

Hierarchy of Privileges: The specific architecture we discussed above assumed two privilege levels. However, it may make sense to have more than two for different use cases. For instance, we may have a privilege level in between those two that allows grouping of *orders* without revealing the PII information. This can be supported through access control on the PII Database; however, there may be alternative approaches that don't require access to the PII Database to run these queries.

Adaptability to Regulations: With rapidly evolving regulations and, more importantly, legal interpretations of those regulations, the decisions that are made may need to be changed. We need to design strategies that support modifying the decisions being made flexibly and efficiently.

7 CONCLUSIONS

In light of the increasing desire to understand and regulate how personal data is collected, used, and shared, we believe the data management community has a crucial role to play in redesigning the data management systems from the ground up to support privacy by design and default. We presented our vision of a redesigned relational database management system that transparently uses pseudonymization, data partitioning, and synthetic data generation, to support the goals of data minimization, access minimization, and consumer rights. Our goal is for this system to serve as a drop-in replacement to existing relational database deployments, but there are still many hard research challenges in fully realizing that vision. We believe many other aspects of how data management systems are architected and built can and need to be similarly revisited to better protect privacy.

REFERENCES

[1] Lindsey Allen, Panagiotis Antonopoulos, Arvind Arasu, Johannes Gehrke, Joachim Hammer, James Hunter, Raghav Kaushik, Donald Kossmann, Jonathan Lee, and Ravi Ramamurthy. 2019. Veritas: Shared verifiable databases and tables in the cloud. In *9th Biennial Conference on Innovative Data Systems Research (CIDR)*.

[2] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security with Cipherbase.. In *CIDR*.

[3] Arvind Arasu, Ken Eguro, Raghav Kaushik, and Ravishankar Ramamurthy. 2014. Querying encrypted data. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1259–1261.

[4] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. [n. d.]. SAQE: Practical Privacy-Preserving Approximate Query Processing for Data Federations. *Proceedings of the VLDB Endowment* 13, 11 ([n. d.]).

[5] Jan Camenisch and Anja Lehmann. 2015. (Un) linkable pseudonyms for governmental databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1467–1479.

[6] Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. 2013. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports* 3 (2013), 1376.

[7] Amol Deshpande and Ashin Machanavajjhala. 2018. Privacy Challenges in the Post-GDPR World: A Data Management Perspective. (2018). http://wp.sigmod.org/?p=2554

[8] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407. https://doi.org/10.1561/0400000042

[9] Peeyush Gupta, Yin Li, Sharad Mehrotra, Nisha Panwar, Shantanu Sharma, and Sumaya Almanee. 2019. Obscure: Information-theoretic oblivious and verifiable aggregation queries. *Proceedings of the VLDB Endowment* 12, 9 (2019), 1030–1043.

[10] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 216–227.

[11] Mike Hintze and Khaled El Emam. 2018. Comparing the benefits of pseudonymisation and anonymisation under the GDPR. *Data Protection and Privacy* (2018).

[12] Noah Johnson, Joseph P Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proceedings of the VLDB Endowment* 11, 5 (2018), 526–539.

[13] Daniel Kifer and Ashwin Machanavajjhala. 2011. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 193–204.

[14] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1371–1384.

[15] Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner. 2019. SchengenDB: A Data Protection Database Proposal. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 24–38.

[16] Anja Lehmann. 2019. ScrambleDB: Oblivious (Chameleon) Pseudonymization-as-a-Service. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 289–309.

[17] Erika McCallister. 2010. *Guide to protecting the confidentiality of personally identifiable information*. Vol. 800. Diane Publishing.

[18] Sharad Mehrotra, Shantanu Sharma, Jeffrey D Ullman, Dhrubajyoti Ghosh, Peeyush Gupta, and Anurag Mishra. 2020. PANDA: Partitioned Data Security on Outsourced Sensitive and Non-sensitive Data. *ACM Transactions on Management Information Systems (TMIS)* 11, 4 (2020), 1–41.

[19] A. Narayanan and V. Shmatikov. 2008. Robust De-anonymization of Large Sparse Datasets. In *IEEE Symposium on Security and Privacy*.

[20] Thomas Neubauer and Johannes Heurix. 2011. A methodology for the pseudonymization of medical data. *International journal of medical informatics* 80, 3 (2011), 190–204.

[21] Primal Pappachan, Roberto Yus, Sharad Mehrotra, and Johann-Christoph Freytag. 2020. Sieve: A Middleware Approach to Scalable Access Control for Database Management Systems. 13, 12 (2020), 2424–2437.

[22] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 85–100.

[23] Jennie Rogers, Johes Bater, Xi He, Ashwin Machanavajjhala, Madhav Suresh, and Xiao Wang. 2019. Privacy Changes Everything. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 96–111.

[24] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 38–54.

[25] Malte Schwarzkopf, Eddie Kohler, M Frans Kaashoek, and Robert Morris. 2019. Position: GDPR compliance by construction. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 39–53.

[26] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. 2020. Understanding and Benchmarking the Impact of GDPR on Database Systems. *PVLDB* 13, 7 (2020), 1064–1077.

[27] Sarah Spiekermann. 2012. The challenges of privacy by design. *Commun. ACM* 55, 7 (2012), 38–40.

[28] Michael Stonebraker, Dong Deng, and Michael L Brodie. 2016. Database decay and how to avoid it. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 7–16.

[29] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–18.

[30] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2015. IntegriDB: Verifiable SQL for outsourced databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1480–1491.