2017 IEEE Symposium on Security and Privacy

SoK: Cryptographically Protected Database Search

Benjamin Fuller*, Mayank Varia†, Arkady Yerukhimovich‡, Emily Shen‡, Ariel Hamlin‡,
Vijay Gadepally‡, Richard Shay‡, John Darby Mitchell‡, and Robert K. Cunningham‡

*University of Connecticut
Email: benjamin.fuller@uconn.edu

†Boston University
Email: varia@bu.edu

‡MIT Lincoln Laboratory
Email: {arkady, emily.shen, ariel.hamlin, vijayg, richard.shay, mitchelljd, rkc}@ll.mit.edu

***Abstract*—Protected database search systems cryptographically isolate the roles of reading from, writing to, and administering the database. This separation limits unnecessary administrator access and protects data in the case of system breaches. Since protected search was introduced in 2000, the area has grown rapidly; systems are offered by academia, start-ups, and established companies.**

However, there is no best protected search system or set of techniques. Design of such systems is a balancing act between security, functionality, performance, and usability. This challenge is made more difficult by ongoing database specialization, as some users will want the functionality of SQL, NoSQL, or NewSQL databases. This database evolution will continue, and the protected search community should be able to quickly provide functionality consistent with newly invented databases.

At the same time, the community must accurately and clearly characterize the tradeoffs between different approaches. To address these challenges, we provide the following contributions:

1) **An identification of the important primitive operations across database paradigms. We find there are a small number of *base* operations that can be used and combined to support a large number of database paradigms.**
2) **An evaluation of the current state of protected search systems in implementing these base operations. This evaluation describes the main approaches and tradeoffs for each base operation. Furthermore, it puts protected search in the context of unprotected search, identifying key gaps in functionality.**
3) **An analysis of attacks against protected search for different base queries.**
4) **A roadmap and tools for transforming a protected search system into a protected database, including an open-source performance evaluation platform and initial user opinions of protected search.**

***Index Terms*—searchable symmetric encryption, property preserving encryption, database search, oblivious random access memory, private information retrieval**

I. Introduction

The importance of collecting, storing, and sharing data is widely recognized by governments [1], companies [2], [3], and individuals [4]. When these are done properly, tremendous value can be extracted from data, enabling better decisions, improved health, economic growth, and the creation of entire industries and capabilities.

Important and sensitive data are stored in database management systems (DBMSs), which support ingest, storage, search, and retrieval, among other functionality. DBMSs are vital to most businesses and are used for many different purposes. We distinguish between the core *database*, which provides mechanisms for efficiently indexing and searching over dynamic data, and the *DBMS*, which is software that accesses data stored in a database. A database's primary purpose is efficient storage and retrieval of data. DBMSs perform many other functions as well: enforcing data access policies, defining data structures, providing external applications with strong transaction guarantees, serving as building blocks in complex applications (such as visualization and data presentation), replicating data, integrating disparate data sources, and backing up important sources. Recently introduced DBMSs also perform analytics on stored data. We concentrate on the database's core functions of data insertion, indexing, and search.

As the scale, value, and centralization of data increase, so too do security and privacy concerns. There is demonstrated risk that the data stored in databases will be compromised. Nation-state actors target other governments' systems, corporate repositories, and individual data for espionage and competitive advantages [5]. Criminal groups create and use underground markets to buy and sell stolen personal information [6]. Devastating attacks occur against government [7] and commercial [8] targets.

Protected database search technology cryptographically separates the roles of providing, administering, and accessing data. It reduces the risks of a data breach, since the server(s) hosting the database can no longer access data contents. Whereas most traditional databases require the server to be able to read all data contents in order to perform fast search and retrieval, protected search technology uses cryptographic techniques on data that is encrypted or otherwise encoded, so that the server can quickly answer queries without being able to read the plaintext data.

This material is based upon work supported under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Air Force. The work of B. Fuller was performed in part while at MIT Lincoln Laboratory. The work of M. Varia was performed under NSF Grant No. 1414119 and additionally while a consultant at MIT Lincoln Laboratory.

DOI 10.1109/SP.2017.10

A. Protected Search Systems Today

Protected database search has reached an inflection point in maturity. In 2000, Song, Wagner, and Perrig provided the first scheme with communication proportional to the description of the query and the server performing (roughly) a linear scan of the encrypted database [9]. Building on this, the field quickly moved from theoretical interest to the design and implementation of working systems.

Protected database search solutions encompass a wide variety of cryptographic techniques, including property-preserving encryption [10], searchable symmetric encryption [11], private information retrieval by keyword [12], and techniques from oblivious random access memory (ORAM) [13]. Like the cryptographic elements used in their construction, protected search systems provide provable security based on the hardness of certain computational problems. Provable security comes with several other benefits: a rigorous definition of security, a thorough description of protocols, and a clear statement of assumptions.

Many of these systems have been implemented. Protected search implementations have been tested and found to scale moderately well, reporting performance results on datasets of billions of records [14]–[22].

In the commercial space, a number of established and startup companies offer products with protected search functionality, including Bitglass [23], Ciphercloud [24], CipherQuery [25], Crypteron [26], IQrypt [27], Kryptnostic [28], Google's Encrypted BigQuery [29], Microsoft's SQL Server 2016 [30], Azure SQL Database [31], PreVeil [32], Skyhigh [33], StealthMine [34], and ZeroDB [35]. While not all commercial systems have undergone thorough security analysis, their core ideas come from published work. For this reason, this paper focuses on systems with a published description.

Designing a protected search system is a balance between security, functionality, performance, and usability. Security descriptions focus on the information that is revealed, or *leaked*, to an attacker with access to the database server. Functionality is primarily characterized by the query types that a protected database can answer. Queries are usually expressed in a standard language such as the structured query language (SQL). Performance and usability are affected by the database's data structures and indexing mechanisms, as well as required computational and network cost.

There are a wide range of protected database systems that are appropriate for different applications. With such a range of choices, it is natural to ask: Are there solutions for every database setting? If so, which solution is best?

B. Our Contribution

The answers to these questions are complex. Protected search replicates the functionality of some database paradigms, but the unprotected database landscape is diverse and rapidly changing. Even for database paradigms with mature protected search solutions, making an informed choice requires understanding the tradeoffs.

The goal of this work is twofold: first, to inform protected search designers on the current and future state of database technology, enabling focus on techniques that will be useful in future DBMSs, and second, to help security and database experts understand the tradeoffs between protected search systems so they can make an informed decision about which technology, if any, is most appropriate for their setting.

We accomplish these goals with the following contributions:

1) A characterization of database search functionality in terms of base and combined queries. Traditional databases efficiently answer a small number of queries, called a *basis*. Other queries are answered by combining these basis operations [36]. Protected search systems have implicitly followed this *basis* and *combination* approach.
   Although there are many database paradigms, the number of distinct bases of operations is small. We advocate for explicitly adopting this basis and combination approach.
2) An identification of the bases of current protected search systems and black-box ways to combine basis queries to achieve richer functionality. We then put protected search in the context of unprotected search by identifying basis functions currently unaddressed by protected search systems.
3) An evaluation of current attacks that exploit *leakage* of various protected search approaches to learn sensitive information. This gives a snapshot of the current security of available base queries.
4) A roadmap and tools for transforming a protected search system into a protected DBMS capable of deployment. We present an open-source software package developed by our team that aids with performance evaluation; our tool has evaluated protected search at the scale of 10TB of data. We also present preliminary user opinions of protected search. Lastly, we summarize systems that have made the transition to full systems, and we challenge other designers to think in terms of full DBMS functionality.

C. Organization

The remainder of this work is organized as follows: Section II introduces background on databases and protected search systems, Section III describes protected search base queries and leakage attacks against these queries, Section IV describes techniques for combining base queries and discusses remaining functionality gaps, Section V shows how to transform from queries to a full system, and Section VI concludes.

II. Overview of Database Systems

This section provides background on the databases and protected search systems that we study in this paper. We first describe unprotected database paradigms and their query bases. Next we define the types of users and operations of a database. We then describe the protected search problem, including its security goals and the security imperfections

known as leakage that schemes may exhibit. Finally, we give examples of common leakage functions found in the literature.

A. Database Definition and Evolution

A database is a partially-searchable, dynamic data store that is optimized to respond to targeted queries (e.g., those that return less than 5% of the total data). Database servers respond to queries through a well established API. Databases typically perform search operations in time sublinear in the database size due to the use of parallel architectures or data structures such as binary trees and B-trees.

Several styles of database engines have evolved over the past few decades. Relational or SQL-style databases dominated the database market from the 1970s to the 1990s. Over the past decade, there has been a focus on databases systems that support many sizes of data management workloads [37]. NoSQL and NewSQL have emerged as new database paradigms, gaining traction in the database market [38], [39].

1) SQL: Relational databases (often called SQL databases) typically provide strong transactional guarantees and have a well known interface. Relational databases are vertically scalable: they achieve better performance through greater hardware resources. SQL databases comply with ACID (Atomicity, Consistency, Isolation, and Durability) requirements [40].

2) NoSQL: NoSQL (short for "not only SQL") databases emerged in the mid 2000s. NoSQL optimizes the architecture for fast data ingest, flexible data structures, and relaxed transactional guarantees [41]. These changes were made to accommodate increasing amounts of unstructured data. NoSQL databases, for the most part, excel at horizontal scaling and when data models closely align with future computation.

3) NewSQL: NewSQL systems bring together the scalability of NoSQL databases and the transactional guarantees of relational databases [42]. Several NewSQL variants are being developed, such as in-memory databases that closely resemble the data models and programming interface of SQL databases, and array data stores that are optimized for numerical data analysis.

4) Future Systems: We expect the proliferation of customized engines that are tuned to perform a relatively small set of operations efficiently. While these systems will have different characteristics, we believe that each system will efficiently implement a small set of basis operations. There are several federated or polystore systems being developed [43]–[45].

The heterogeneous nature of current and future databases demands a variety of protected search systems. While providing such variety is challenging, there are a small number of base operations that can be combined to provide much of the functionality of the aforementioned systems.

B. Query Bases

To reduce the space of possible queries that must be secured, we borrow an approach from developers of software specifications and mathematical libraries [46]. In these fields, it is common to determine a core set of low-level kernels and then express other operations using these kernels. Similarly, many database technologies have a *query basis*: a small set of *base* operations that can be *combined* to provide complex search functionality. Furthermore, multiple technologies share the same query basis. In some cases the basis was not explicit in the original design but was formalized in later work. Apache Accumulo's native API does not have a rigorous mathematical design, but frameworks such as D4M [47], [48] and Pig [49] used to manipulate data in Accumulo do.

Leveraging an underlying query basis will allow the protected search community to keep pace with new database systems. We discuss three bases found in database systems. First, relational algebra forms the backbone of many SQL and NewSQL systems [42]. Second, associative arrays provide a mathematical basis for SQL, NoSQL, and NewSQL systems [50]. Third, linear algebraic operations form a basis for some NewSQL databases. These bases and database paradigms are summarized in Table I.

1) Relational Algebra: Relational algebra, proposed by Codd in 1970 as a model for SQL [36], consists of the following primitives: set union, set difference, Cartesian product (joins), projection, and selection. Complex queries can typically be generated by composing these operations. Relational algebra and the composability of operations allow a server-side query planner to optimize query execution by rearranging operations to still produce the same result [68].

2) Associative Arrays: Associative arrays are a mathematical basis for several styles of database engines [50]. They provide a mathematical foundation for key-value store NoSQL databases. Associative array algebra consists of the following base operations: construction, find, associative array addition, associative array element-wise multiplication, and array multiplication [47]. Associative arrays are built on top of the algebraic concept of a semiring (a ring without an additive inverse). Addition or multiplication in an associative array can denote any two binary operations from an algebraic semiring. Usually, these two operations are the traditional $\times$ and $+$, but in the min-plus algebra the two operations are min and $+$ (in the max-plus algebra the two operations are max and $+$).

3) Linear Algebra: A number of newer NewSQL databases support linear algebraic operations. GraphBLAS is a current standardization effort underway for graph algorithms [69]. In GraphBLAS, graph data is stored using sparse matrices, and the linear algebraic base operations of construction, find, matrix addition, matrix multiplication, and element-wise multiplication are composed to create graph algorithms. Examples of how the GraphBLAS can be applied to popular graph algorithms are given in [70], [71].

C. Database Roles and Operations

We consider five important database *roles*, analogous to roles in database systems like Microsoft SQL Server 2016 [72]:

- A **provider**, who provides and modifies the data.
- A **querier**, who wishes to learn things about the data.
- A **server**, who handles storage and processing.

Query Basis	Technology	Fundamental characteristics	Strengths	Weaknesses	Examples
Rel. Algebra Set Union Set Difference Products/Joins Projection Selection	**SQL** [36]: Relational	Transaction support, ACID guarantees, Table representation of data	Popular interface, Common data model [51]	Upfront schema design, Low insert & query rate	MySQL [52] Oracle DB [53] Postgres [54]
	NewSQL [42]: Relational	Use of in-memory, new system arch. or simplified data model	Popular interface, Transactional support, ACID guarantees	Req. expensive hardware, Often only relational data model	Spanner [55] MemSQL [56] Spark SQL [57]
	Federated [58]	Relational model, Partitioned/replicated tables	Transactional support, High performance, ACID guarantees	Upfront schema design, Often only relational data model	Garlic [58] DB2 [59]
Assoc. Array Alg. Construct Find Array $(+, \times)$ Element-wise $\times$	**NoSQL** [41]: Key-value	Horizontal scalability, Data rep. as key-value pairs, BASE guarantees [60]	High insert rates, Cell-level security Flexible schema	Sacrifice one of the following: consistency, availability, or partition tolerance	BigTable [41] Accumulo [61] HBase [62] mongoDB [63]
Linear Algebra Construct Find Matrix $(+, \times)$ Element-wise $\times$	**NoSQL** [64]: Graph Databases	Data represented as adjacency or incidence matrix, Horizontal scalability, Graph operation API	Natural data representation, Amenable to graph algs.	Performance, Diverse data models, Difficult to optimize	Neo4j [64] System G [65]
	NewSQL [66]: Array Databases	ACID guarantees, Data represented as arrays (dense or sparse)	High performance, Transactional support, Good for scientific comp.	Data model restrictions, Lack of iterator support	SciDB [66] TileDB [67]
Multiple bases	**Polystore** [43]	Disparate DBMSs	High performance, Flexible data stores, Diverse data/programming models	Requires middleware	BigDAWG [43] Myria [45]

TABLE I

SUMMARY OF A (NOT EXHAUSTIVE) SET OF POPULAR CURRENT AND EMERGING QUERY BASES TOGETHER WITH THEIR CORRESPONDING DATABASE TECHNOLOGIES. CHARACTERISTICS, STRENGTHS, WEAKNESSES, AND EXAMPLES REFER TO THE TECHNOLOGIES, NOT THE QUERY BASES.

- An **authorizer**, who specifies data- and query-based rules.
- An **enforcer**, who ensures that rules are applied.

Databases provide an expressive language for representing permissions, or *rules*. Rules are enforced by authenticating the roles possessed by a valid *user* and granting her the appropriate powers. In general, each user may perform multiple roles, and each role may be performed by multiple users.

While databases offer a wide range of features, we focus on four operations: **Init**, **Query**, **Update**, and **Refresh**. These operations are common across the database paradigms described above; we describe them below in the context of protected search.

- **Init**: The initialization protocol occurs between the provider (who has data to load) and the server. The server obtains a protected database representing the loaded data.
- **Query**: The query protocol occurs between the querier (with a query), the server (with the protected database), the enforcer (with the rules), and possibly the provider. The querier obtains the query results if the rules are satisfied.
- **Update**: The update protocol occurs between the provider (with a set of updates) and the server. The server obtains an updated protected database. Updates include insertions, deletions, and record modifications.
- **Refresh**: The refresh protocol occurs between the provider and the server. The server obtains a new protected database that represents the same data but is designed to achieve better performance and/or security.

All systems considered in this work support **Init** and **Query**, but only some systems support **Update** and **Refresh**; see Tables II and V for details.

D. Protected Database Search Systems

Informally, a protected search system is a database system that supports the roles and operations defined above, in which each party learns only its intended outputs and nothing else. In particular, a protected search system aims to ensure that the server learns nothing about the data stored in the protected database or about the queries, and the querier learns nothing beyond the query results. These security goals can be formalized using the *real-ideal* style of cryptographic definition. In this paradigm, one imagines an ideal version of a protected search system, in which a trusted external party performs storage, queries, and modifications correctly and reveals only the intended outputs to each party. The real system is said to be secure if no party can learn more from its real world interactions than it can learn in the ideal system.

We restrict our attention in this work to protected database search systems that provide formally defined security guarantees based upon the strength of their underlying cryptographic primitives. Some of the commercial systems mentioned in the introduction lack formal security reductions; although they are based on techniques with proofs of security, analysis is required to determine whether differences from those proven techniques affect security.

Scenarios: Only a few existing protected search systems consider the enforcement of rules (i.e., include an authorizer and enforcer). Therefore, in this paper we focus primarily on two scenarios: a *three-party* scenario comprising a provider, a querier, and a server, and a *two-party* scenario in which a single user acts as both the provider and the querier (we denote

this combined entity as the *client*). The latter scenario models a cloud storage outsourcing application in which a client uploads files to the cloud that she can later request. In the two-party setting, the client has the right to know all information in the database so it is only necessary to consider security against an adversarial server. In this work, we focus on protected search in the case of a single provider and a single querier; for the more general setting in which multiple users can perform a single role, see Section V and [73].

We stress that a secure search system for one scenario does not automatically extend to another scenario. Additionally, despite the limited attention in the literature thus far, we believe that the authorizer and enforcer roles are an important aspect of the continued maturation of protected search systems; see Section V-A for additional discussion.

Threats: There are two types of entities that may pose security threats to a database: a valid user known as an *insider* who performs one or more of the roles, and an *outsider* who can monitor and potentially alter network communications between valid users. We distinguish between adversaries that are *semi-honest* (or *honest-but-curious*), meaning they follow the prescribed protocols but may passively attempt to learn additional information from the messages they observe, and those that are *malicious*, meaning they are actively willing to perform any action necessary to learn additional information or influence the execution of the system. An outsider adversary (even a malicious one) can be thwarted using secure point-to-point channels. Furthermore, we distinguish between adversaries that *persist* for the lifetime of the database and those that obtain a *snapshot* at a single point in time [74]. The bulk of active research in protected search technology considers semi-honest security against a persistent insider adversary.

Performance and Leakage: While unprotected databases are often I/O bound, protected systems may be compute or network bound. We can measure the performance of a protected operation by calculating the computational overhead and the additional network use (in both the number of messages and the total amount of data transmitted). The type of cryptographic operations matters as well: whenever possible, slower public-key operations should be avoided or minimized in favor of faster symmetric-key primitives.

In order to improve performance, many protected search systems reveal or *leak* information during some or all operations. Leakage should be thought of as an imperfection of the scheme. The real-ideal security definition is parameterized by the system's specific *leakage profile*, which comprises a sequence of functions that formally describe all information that is revealed to each party beyond the intended output. A security proof demonstrates that the claimed leakage profile is an upper bound on what is actually revealed to an adversary. Protected search systems' security is primarily distinguished by their leakage profile; our security discussion focuses on leakage.

While leakage profiles are comprehensive, it is often difficult to interpret them and to assess their impact on the security of a particular application (see Section III-B). To help with this task, the next section identifies common types of leakage.

E. Common Leakage Profiles

This section provides a vocabulary (partially based on Kamara [75]) to describe common features of leakage systematically. While the exact descriptions of leakage profiles are often complex, their essence can mostly be derived from four characteristics: the objects that leak, the type of information leaked about them, which operation leaks, and the party that learns the leakage.

The following types of objects within a protected search system are vulnerable to leakage.

1) Data items, and any indexing data structures.
2) Queries.
3) Records returned in response to queries, or other relationships between the data items and the queries (e.g., records that partially match a conjunction query).
4) Access control rules and the results of their application.

Next, we categorize the information leaked from each object. The leakage may occur independently for each query or response, or it may depend upon the prior history of queries and responses. For complex queries like Booleans, leakage may also depend on the connections between the clauses of a query. While the details of leakage may depend on the specific data structures used for representing and querying the data, we list five general categories of information that may be leaked from objects, ranked from the least to most damaging. We use this ranking throughout our discussion of base queries.

- ○ Structure: properties of an object only concealable via padding, such as the length of a string, the cardinality of a set, or the circuit or tree representation of an object.
- ◔ Identifiers: pointers to objects so that their past/future accesses are identifiable.
- ◐ Predicates: identifiers plus additional information on objects. Examples include "matches the intersection of 2 clauses within a query" and "within a common (known) range."
- ◕ Equalities: which objects have the same value.
- ● Order (or more): numerical or lexicographic ordering of objects, or perhaps even partial plaintext data.

Each of the four database operations may leak information. During **Init**, the server may receive leakage about the initial data items. Every party may receive leakage during a **Query**: the querier may learn about the rules and the current data items; the server may learn about the query, the rules, and the current data items; the provider may learn about the query and rules; and the enforcer may learn about the query and current data items. During **Update**, the server may receive leakage about the prior and new data records. During a **Refresh**, the server may receive leakage about the current data items.

In a two-party protected search system without **Update** or rules it suffices to describe the leakage to the server during **Init** and **Query**. In this setting, common components of leakage profiles include: equalities of queries (often called *search patterns*); identifiers of data items returned across multiple

queries (often called *access patterns*); the circuit topology of a boolean query; and cardinalities and lengths of data items, queries, and query responses. Dynamic databases must also consider leakage during **Update** and **Refresh**. Three-party databases with access restrictions must also consider leakage to the provider and querier about any objects they didn't produce themselves.

F. Comparison with Other Approaches

We intentionally define protected database search by its objective rather than the techniques used. As we will see in Section III, many software-based techniques suffice to construct protected database search. Many hardware-based solutions like [76] are viable and valuable as well; however, they use orthogonal assumptions and techniques to software-only approaches. To maintain a single focus in this SoK, we restrict our attention to software-only approaches.

Within software-only approaches, the cryptographic community has developed several general primitives that address all or part of the protected database search problem.

- Secure multi-party computation [77]–[79], fully homomorphic encryption [80]–[82], and functional encryption [83] hide data while computing queries on it.
- Private information retrieval [12], [84], [85] and oblivious random-access memory (ORAM) [13] hide access patterns over the data retrieved. On their own, they typically do not support searches beyond a simple index retrieval; however, several schemes we discuss in the next section use ORAM in their protocols to hide access patterns while performing expressive database queries.

Protected search techniques in the literature often draw heavily from these primitives, but rarely rely exclusively on one of them in its full generality. Instead, they tend to use specialized protocols, often with some leakage, with the goal of improving performance.

Another related area of research known as authenticated data structures ensures correctness in the presence of a malicious server but does not provide confidentiality (e.g., [86]–[90]). In general, authenticated data structures do not easily compose with protected database search systems.

III. Base Queries

In this section, we identify basis functions that currently exist in protected search. The section provides systematic reviews of the different cryptographic approaches used across query types and an evaluation of known attacks against them.

Due to length limitations, we focus on the *Pareto frontier* of schemes providing the currently best available combinations of functionality, performance, and security. This means that we omit any older schemes that have been superseded by later work. For a historical perspective including such schemes, we refer readers to relevant surveys [73], [91].

We categorize the schemes into three high-level approaches. The Legacy Index (or `Legacy`) approach can be used with an unprotected database server; it merely modifies the provider's data insertions and the querier's requests. However, this backwards compatibility comes at a cost to security. The Custom Index (or `Custom`) approach achieves lower leakage at the expense of designing special-purpose protected indices together with customized protocols that enable the querier and server to traverse the indices together. We highlight a third approach Oblivious Index (or `Obliv`), which is a subset of `Custom` that provides stronger security by obscuring object identifiers (i.e., hiding repeated retrieval of the same record).

A. Base Query Implementations

Cryptographic protocols have been developed for several classes of base queries. The most common constructions are for equality, range, and boolean queries (which evaluate boolean expressions over equality and/or range clauses), though additional query types have been developed as well. Here, we summarize some of the techniques for providing these functionalities, splitting them based on the approach used.

The text below focuses on the distinct benefits of each base query mechanism; Table II systematizes the common security, performance, and usability dimensions along which each scheme can be measured. From a security point of view, we list the index approach, threat model (cf. Section II-D), and the amount of leakage that the server learns about the data items during **Init** and **Query** (cf. Section II-E). Performance and usability are described along three dimensions: the *scale* of updates and queries that each scheme has been shown to support, the type and amount of *cryptography* required to support updates and queries, and the *network* latency and bandwidth characteristics.

1) `Legacy`*:* Property-preserving encryption [10] produces ciphertexts that preserve some property (e.g., equality or order) of the underlying plaintexts. Thus, protected searches (e.g., equality or range queries) can be supported by inserting ciphertexts into a traditional database, without changing the indexing and querying mechanisms. As a result, `Legacy` schemes immediately inherit decades of advances and optimizations in database management systems.

Equality: Deterministic encryption (DET) [15], [92] applies a randomized-but-fixed permutation to all messages so equality of ciphertexts implies equality of plaintexts, enabling lookups over encrypted data. All other properties are obscured. However, deterministic encryption typically reveals equalities between data items to the server even without the querier making any queries.

Range: Order-preserving encryption (OPE) [93]–[95] preserves the relative order of the plaintexts, enabling range queries to be performed over ciphertexts. This approach requires no changes to a traditional database, but comes at the cost of quite significant leakage: roughly, in addition to revealing the order of data items, it also leaks the upper half of the bits of each message [94]. Improving on this, Boldyreva et al. [95] show how to hide message contents until queries are made against the database. Mavroforakis et al. [96] further strengthen security using fake queries. Finally, mutable

OPE [97] reveals only the order of ciphertexts at the expense of added interactivity during insertion and query execution.

Many Legacy approaches can easily be extended to perform boolean queries and joins by simply combining the results of the equality or range queries over the encrypted data. CryptDB [15] handles these query types using a layered or *onion* approach that only reveals properties of ciphertexts as necessary to process the queries being made. They demonstrate at most 30% performance overhead over MySQL, though this value can be much smaller depending on the networking and computing characteristics of the environment.

Legacy approaches have been adopted industrially [98] and deployed in commercial systems [23]–[35]. However, as we will explain in Section III-B and Table III, even the strongest Legacy schemes reveal substantial information about queries and data to a dedicated attacker.

2) Custom *Inverted Index:* Several works over the past decade support equality searches on single-table databases via a reverse lookup that maps each keyword to a list of identifiers for the database records containing the keyword (e.g., [11], [99]). Newer works provide additional features and optimizations for such equality searches. Blind Storage [100] shows how to do this with low communication and a very simple server, while Sophos [101] shows how to achieve a notion of forward security hiding whether new records match older queries (this essentially runs **Refresh** on every **Insert**).

OSPIR-OXT [18]–[21] additionally supports boolean queries: the inverted index finds the set of records matching the first term in a query, and a second index containing a list of (record identifier, keyword) pairs is used to check whether the remaining terms of the query are also satisfied. Cryptographically, the main challenge is to link the two indices obliviously, so that the server only learns the connections between terms in the same query. Going beyond boolean queries, Kamara and Moataz [102] intelligently combine several inverted indices in order to support the selection, projection, and Cartesian product operations of relational algebra with little overhead on top of the underlying inverted index (specifically, only using symmetric cryptography). They do so at the expense of introducing additional leakage. Moataz's Clusion library implements many inverted index-based schemes [103], [104].

Cash and Tessaro demonstrate that secure inverted indices must necessarily be slower than their insecure counterparts, requiring extra storage space, several non-local read requests, or large overall information transfer [105].

3) Custom *Tree Traversal:* Another category of Custom schemes uses indices with a tree-based structure. Here a query is executed (roughly) by traversing the tree and returning the leaf nodes at which the query terminates. The main cryptographic challenge here is to hide the traversal pattern through the tree, which can depend upon the data and query.

For equality queries, Kamara and Papamanthou [106] show how to do this in a parallelizable manner; with enough parallel processing they can achieve an amortized constant query cost. Stefanov et al. [107] show how to achieve forward privacy using a similar approach.

The BLIND SEER system [16], [17] supports boolean queries by using an index containing a search tree whose leaves correspond to records in the database, and whose nodes contain (encrypted) Bloom filters storing the set of all keywords contained in their descendants. A Bloom filter is a data structure that allows for efficient set membership queries. To execute a conjunctive query, the querier and server jointly traverse the tree securely using Yao's garbled circuits [108], a technique from secure two-party computation, following branches whose Bloom filters match all terms in the conjunction. Chase and Shen [109] design a protection method based on suffix trees to enable substring search.

Tree-based indices are also amenable to range searches. The Arx-RANGE protocol [110] builds an index for answering range queries without revealing all order relationships to the server. The index stores all encrypted values in a binary tree so range queries can be answered by traversing this tree for the end points. Using Yao's garbled circuits, the server traverses the index without learning the values it is comparing or the result of the comparison at each stage. Roche et al.'s partial OPE protocol [111] provides a different tradeoff between performance and security with a scheme optimized for fast insertion that achieves essentially free insertion and (amortized) constant time search at the expense of leaking a partial order of the plaintexts.

4) Other Custom *Indices:* We briefly mention protected search mechanisms supporting other query types: ranking results of boolean queries [112], [113], calculating the inner product with a fixed vector [114], [115], and computing the shortest distance on a graph [116]. These schemes mostly work by building encrypted indices out of specialized data structures for performing the specific query computation. For example, Meng et al.'s GRECS system [116] provides several different protocols with different leakage/performance tradeoffs that encrypt a sketch-based (graph) distance oracle to enable secure shortest distance queries.

5) Obliv*:* This class of protected search schemes aims to hide common results between queries. Oblivious RAM (ORAM) has been a topic of research for twenty years [117] and the performance of ORAM schemes has progressed steadily. Many of the latest implementations are based on the Path ORAM scheme [118]. However, applying ORAM techniques to protected search is still challenging [119].

Obliv schemes typically hide data identifiers across queries by re-encrypting and moving data around in a data structure (e.g., a tree) stored on the server. Several equality schemes use the Obliv approach. Roche et al.'s vORAM+HIRB scheme [120] observes that search requires an ORAM capable of storing varying size blocks since different queries may result in different numbers of results. They design an efficient variable-size ORAM (vORAM) and combine it with a history independent data structure to build a keyword search scheme. Garg et al.'s TWORAM scheme [121] focuses on reducing the number of rounds required by an ORAM-type secure search. They use a garbled RAM-like [122] construction to build a two-round ORAM resulting in a four-round search scheme

for equality queries. Moataz and Blass [123] design oblivious versions of suffix arrays and suffix trees to provide an `Obliv` scheme for substring queries. While offering greater security, these schemes still tend to be slower than the constructions in the other classes.

An alternative approach is to increase the number of parties. This approach is taken by Faber et al.'s 3PC-ORAM scheme [124] and Ishai et al.'s shared-input shared-output symmetric private information retrieval (SisoSPIR) protocol [22] to support range queries. 3PC-ORAM shows how by adding a second non-colluding server, one can build an ORAM scheme that is much simpler than previous constructions. SisoSPIR uses a distributed protocol between a client and two non-colluding servers to traverse a (per-field) B-tree in a way that neither server learns anything about which records are accessed. By deviating from the standard ORAM paradigm, these schemes are able to approach the efficiency typically achieved by Custom Index schemes that do not hide access patterns.

6) Supporting Updates: Another important aspect of secure search schemes is whether they support **Update**. While update functionality is critical for many database applications, it is not supported by many protected search schemes in the `Custom` and `Obliv` categories. Those that support updates do so in one of two ways. For ease of presentation, consider a newly inserted record. In most `Legacy` schemes the new value is immediately inserted into the database index, allowing for queries to efficiently return this value immediately after insertion. In many `Custom` schemes, e.g., [16], new values are inserted into a side index on which a less efficient (typically, linear time) search can be used. Periodically performing **Refresh** incorporates this side index into the main index; however, due to the cost of **Refresh** it is not possible to do this very frequently. Thus, depending on the frequency and size of updates, update capability may be a limiting functionality of protected search. In particular, a major open question is to build protected search capable of supporting the very high ingest rates typical of NoSQL databases. We return to this open problem in Section V. Roche et al. [111] take a step in this direction with a `Custom` scheme for range queries capable of supporting very high insert rates.

Table II systematizes the protected search techniques discussed in this section along with some basic information about the (admittedly nuanced) leakage profiles that they have been proven to meet. There are several correlations between columns of the table; some of these connections reveal fundamental privacy-performance tradeoffs whereas others simply reflect the current state of the art. To provide one example in the latter category: most `Legacy` systems leak information at ingestion, whereas most `Custom` only leak information after queries have been made against the system. The recent Arx-EQ [14] bucks this trend by requiring the client to remember the frequency of each keyword.

B. Leakage Inference Attacks

In this subsection and Table III, we summarize *leakage inference attacks* that can exploit the leakage revealed by a protected search system in order to recover some information about sensitive data or queries. Hence, this section details the real-world *impact* of the leakage bounds and threat models depicted in Table II. The two tables are connected via a JOIN on the "$\mathcal{S}$ leakage" columns: a protected search scheme is affected by an attack if the scheme's leakage to the server is at least as large as the attack's required minimum leakage.

We stress that leakage inference is a new and rapidly evolving field. As a consequence, the attacks in Table III only cover a subset of leakage profiles included in Table II. Additionally, this section merely provides lower bounds on the impact of leakage because attacks only improve over time.

We start by introducing the different dimensions that characterize attack requirements and efficacy. Then, we sketch a couple representative attacks from the literature. Finally, we describe how the provider and querier should use these attacks to inform their choice of a search system that adequately protects their interests.

1) Attack Requirements: We classify attacks along four dimensions: attacker goal, required leakage, attacker model, and prior knowledge. The attacker is the server in all of the attacks we consider, except for the Communication Volume Attack of [125], which can be executed by a network observer who knows the size of the dataset. We expect future research on attacks using leakage available to other insiders.

a) Attacker Goal: Current attacks try to recover either a set of queries asked by the querier (*query recovery*) or the data being stored at the server (*data recovery*).

b) Required Leakage: This is the leakage function that must be available to the attacker. We focus on the common leakage functions on the dataset and responses identified in Section II-E. Examples include the cardinality of a response set, the ordering of records in the database, and identifiers of the returned records. Some attacks require leakage on the entire dataset while others only require leakage on query responses.

c) Attacker Model: Current inference attacks assume one of two attacker models. The first is a *semi-honest* attacker as discussed in Section II-D. The second is an attacker capable of *data injection*: it can create specially crafted records and have the provider insert them into the database. Note that this capability falls outside the usual malicious model for the server. The attacker's ability to perform data injection depends on the use case. For example, if a server can send an email to a user that automatically updates the protected database, this model is reasonable. On the other hand, it might be harder to insert an arbitrary record into a database of hospital medical records.

d) Attacker Prior Knowledge: All current attacks assume some prior knowledge, which is usually information about the stored data but may include information about the queries made. Attack success is judged by the ability to learn information beyond the prior knowledge. The following types of prior

knowledge (ordered from most to least information) help to execute attacks.

- ● Contents of full dataset: the data items contained in the database. The only possible attacker goal in this case is query recovery.
- ◕ Contents of a subset of dataset: a set of items contained in the database. Both attacker goals are interesting in this case.
- ◐ Distributional knowledge of dataset: information about the probability distribution from which database entries are drawn. For example, this could include knowledge of the frequency of first names in English-speaking countries. This type of knowledge can be gained by examining correlated datasets.
- ◔ Distributional knowledge of queries: information about the probability distribution from which queries are drawn. As above, this might be knowledge that names will be queried according to their frequency in the overall population.
- ○ Keyword universe: knowledge of the possible values for each field.

Naturally, attacks that require full knowledge of the data are more effective; the reasonableness of this assumption should be evaluated for each use case.

2) Attack Efficacy: We evaluate attack efficacy qualitatively in terms of three metrics: 1) the runtime of the attack, including time required to create any inserted records; 2) the sensitivity of the recovery rate to the amount of prior knowledge; and 3) the keyword universe size attacked. Note that the strength of an attack is strongly application-dependent; an attack that is devastating on one dataset may be completely ineffective on another dataset.

Table III characterizes currently known attacks based upon their requirements and efficacy. All of the attacks described in the table only require modest computing resources.

3) Attack Techniques: Leakage inference attacks against protected search systems have evolved rapidly over the last few years, with Islam et al. [132] in 2012 inspiring many other papers. Most of the attacks in Table III rely on the following two facts: 1) different keywords are associated with different numbers of records, and 2) most systems reveal keyword identifiers for a record either at rest (e.g., DET [15] reveals during **Init** if records share keywords) or when it is returned across multiple queries (e.g., Blind Seer [16] reveals during **Query** which returned records share keywords). To give intuition for how these attacks work we briefly summarize two entries of Table III.

Cash et al.'s [128] Count Attack is a conceptually simple way to exploit this information. Assume the attacker has full knowledge of the database and is trying to learn the query. The attacker sees how many records are returned in response to a query. If that number is unique it can identify the query. Furthermore, by identifying the query, the attacker learns that every returned record is associated with that keyword.

For example, suppose the attacker learns the first query was for LastName = 'Smith'. Now consider a second query for an unknown first name. The query does not return a unique number of records, so the method above cannot be used. Suppose that FirstName='John' and FirstName='Matthew' both return 1000 records. The attacker can also check how many records are in common with the previous query. This creates an additional constraint, for example there may be 100 records with name 'John Smith' but only 10 records with name 'Matthew Smith'. By checking record overlap with the previously identified query, the attacker can identify the queried first name. This attack iteratively identifies queries and uses them as additional constraints to identify unknown queries.

Cash et al.'s attack is fairly simple and performs well if the keyword universe sizes is at most 5000. However, it requires a large portion of the dataset to be known to the attacker. With 80% of the dataset known to the attacker, Cash et al. [128] yield a 40% keyword recovery rate.

Zhang et al. [127] extend the Count Attack to a malicious adversary setting, allowing a server to inject a set of constructed records. This capability greatly improves keyword recovery. By carefully constructing a small number of these records (e.g., nine records for a universe of 5000 keywords), it is possible to search the keyword universe and identify the keyword. Although the records are fairly large, the attack extends if the database only allows a limited number of keywords per data record. This attack recovers more keywords than the attack of Cash et al.: 40% of the data must be leaked to obtain a 40% keyword recovery rate.

4) Discussion: The provider and querier rely upon protected search to protect themselves against the server, or anyone who compromises the server. Our systemization of attacks shows that they should consider the following four questions before choosing a protected search technique to use.

- How large is the keyword universe?
- How much of the dataset or query keyword universe (and corresponding frequency) can the attacker predict?
- Can an attacker reasonably insert crafted records?
- Does the adversary have persistent access to the server, or merely a snapshot of it at a single point in time?

Answers to the first three questions depend upon the intended use case. For example, a system with a smaller leakage profile may be necessary in a setting where the keyword universe is small and the attacker has the ability to add records. A system with a larger leakage profile may suffice in a setting where the keyword universe is very large.

The fourth question pertains to adversaries who compromise the server. `Legacy` schemes tend to leak information about the entire database to the server. Thus, using the terminology of Grubbs et al. [74], they are susceptible to an adversary who only gets a *snapshot* of the database at some point in time. In contrast, `Custom` schemes tend to reveal information about records only during record retrieval or index modification as part of the querying process, so they require a *persistent* adversary who can observe the evolution of the database state over time. (We note however that many Boolean schemes have additional leakage about data statistics for the entire database.)

Query type	Scheme (References)	Approach	Threats: # of parties	Threats: Adversarial $\mathcal{Q}$	Threats: Adversarial $\mathcal{S}$	$\mathcal{S}$ leakage: Init	$\mathcal{S}$ leakage: Query	Scale: Updatable?	Scale: Implemented?	Scale: Scale tested	Crypto: Crypto type	Crypto: Insert: # ops	Crypto: Query: # ops	Network: # round trips	Network: Data sent	Unique feature
Equality	Arx-EQ [14]	Legacy	2	—	◐	○	◔	●	✔	◐	●	●	●	●	●	legacy compliant
	Kamara-Papamanthou [106]	Custom	2	—	◐	○	◔	●	—	—	●	○	○	●	●	parallelizable
	Blind Storage [100]	Custom	2	—	◐	○	◔	●	✔	◐	●	◐	●	◐	●	low $\mathcal{S}$ work
	Sophos ($\Sigma o \phi o \varsigma$) [101]	Custom	2	—	◐	○	◔	●	✔	◐	○	◐	●	●	●	**Refresh** w/ **Insert**
	Stefanov et al [107]	Custom	2	—	◐	○	◔	●	✔	◐	●	○	○	●	●	**Refresh** w/ **Insert**
	vORAM+HIRB [120]	Obliv	2	—	◐	○	○	●	✔	●	●	○	○	○	◔	history independ.
	TWORAM [121]	Obliv	2	—	◐	○	○	●	—	—	◐	○	○	◐	◔	const round
	3PC-ORAM [124]	Obliv	3	◐	◐	○	○	●	✔	◔	●	○	○	○	◔	dual $\mathcal{S}$
Boolean	DET [15], [92]	Legacy	2	—	◐	◕	◕	●	✔	●	●	◐	◐	●	●	supports JOINs
	BLIND SEER [16], [17]	Custom	3	●	●	○	◐	◐	✔	●	◐	◐	○	○	◔	hide field, r_i's
	OSPIR-OXT [18]–[21], [104]	Custom	3	●	◐	○	◐	●	✔	●	◐	◐	◐	◕	●	excels w/ small r_1
	Kamara-Moataz [102]	Custom	2	—	◐	○	◐	○	—	—	◐	◐	○	●	◔	relational SPC
Range	OPE [93]–[95]	Legacy	2	—	◐	●	●	●	✔	●	●	●	●	●	●	leak some content
	Mutable OPE [97]	Legacy	2	—	◐	●	●	●	✔	◐	●	○	○	○	◐	interactive
	Partial OPE [111]	Custom	2	—	◐	○	●	●	✔	◐	●	●	●	◐	●	fast insertions
	Arx-RANGE [110]	Custom	2	—	◐	○	◐	●	✔	◐	◐	○	○	●	○	non-interactive
	SisoSPIR [22]	Obliv	3	◐	◐	○	○	○	✔	●	●	●	●	○	◔	split, non-colluding $\mathcal{S}$
Other	GraphEnc$_1$ [116]	Custom	2	—	◐	◐	◔	○	✔	◐	●	●	●	●	◔	approx. graph dist.
	GraphEnc$_3$ [116]	Custom	2	—	◐	◐	◐	○	✔	◐	○	●	●	●	●	approx. graph dist.
	Chase-Shen [109], [126]	Custom	2	—	●	○	◐	○	✔	◔	●	●	●	◐	●	substring search
	Moataz-Blass [123]	Obliv	2	—	◐	○	○	●	✔	●	●	○	○	○	◔	substring search

TABLE II

SUMMARY OF THE SECURITY, PERFORMANCE, AND USABILITY OF BASE QUERIES. $\mathcal{Q}$ AND $\mathcal{S}$ DENOTE THE QUERIER AND THE SERVER, RESPECTIVELY. WE PRESUME THAT THE ADVERSARY KNOWS THE DATABASE SIZE d AND THE LENGTH OF EACH RECORD. FOR SYSTEMS THAT EITHER DO NOT SUPPORT INSERT OR USE A SIDE INDEX, THE INSERT COST IS THE AMORTIZED COST OF ADDING A SINGLE RECORD DURING **Init**. LEGENDS FOR EACH COLUMN FOLLOW. IN ALL COLUMNS EXCEPT "INIT/QUERY LEAKAGE," BUBBLES THAT ARE MORE FILLED IN REPRESENT PROPERTIES THAT ARE BETTER FOR THE SCHEME.

SCALE TESTED
- ●– BILLIONS
- ◐– MILLIONS
- ◔– THOUSANDS

UPDATABLE
- ●– INSERT IN MAIN INDEX
- ◐– BUILD SIDE INDEX
- ○– NOT SUPPORTED

THREATS
- ●– MALICIOUS
- ◐– SEMI-HONEST

DATA SENT (BEYOND RESULTS)
- ●– CONSTANT
- ◐– ADDITIVE POLYLOG(d)
- ◔– MULT. POLYLOG(d)
- ○– EVEN MORE

INIT/QUERY LEAKAGE (SEE SECTION II-E)
- ●– ORDER/CONTENTS
- ◕– EQUALITY
- ◐– PREDICATE
- ◔– IDENTIFIER
- ○– STRUCTURE

TYPE OF CRYPTO
- ●– SYMMETRIC
- ◐– BATCHED OR PRE-COMPUTED PUBLIC-KEY
- ○– PUBLIC-KEY

CRYPTO OPS PER RECORD
- ●– CONSTANT
- ◐– # KEYWORDS
- ○– LOGARITHMIC

ROUND TRIPS
- ●– 1
- ◕– 2
- ◐– CONSTANT
- ○– LOGARITHMIC

Attacker goal	Required $\mathcal{S}$ leakage: Init	Required $\mathcal{S}$ leakage: Query	Required attack conditions: Ability to inject data	Required attack conditions: Prior knowledge	Attack efficacy: Runtime	Attack efficacy: Sensitivity to prior knowledge	Attack efficacy: Keyword universe tested	Attack name
Query Recovery	○	○	—	◔	●	?	○	Communication Volume Attack [125]
	○	◔	✔	○	○	○	○	Binary Search Attack [127]
	○	◔	—	◔	●	?	○	Access Pattern Attack [125]
	○	◔	—	◕	◐	●	●	Partially Known Documents [128]
	○	◔	✔	◕	◐	○	●	Hierarchical-Search Attack [127]
	○	◔	—	●	◐	●	●	Count Attack [128]
Data Recovery	○	◔	—	◐	●	●	◐	Graph Matching Attack [129]
	◕	—	—	◐	○	?	○	Frequency Analysis [130]
	◕	—	✔	◐	○	?	●	Active Attacks [128]
	◕	—	—	◕	○	?	●	Known Document Attacks [128]
	●	—	—	◐	○	○	●	Non-Crossing Attack [131]

TABLE III

SUMMARY OF CURRENT LEAKAGE INFERENCE ATTACKS AGAINST PROTECTED SEARCH BASE QUERIES. $\mathcal{S}$ IS THE SERVER AND THE ASSUMED ATTACKER FOR ALL ATTACKS LISTED. $\mathcal{S}$ LEAKAGE SYMBOLS HAVE THE SAME MEANING AS IN TABLE II. EACH ATTACK IS RELEVANT TO SCHEMES IN TABLE II WITH AT LEAST THE $\mathcal{S}$ LEAKAGE SPECIFIED IN THIS TABLE. SOME ATTACKS REQUIRE THE ATTACKER TO BE ABLE TO INJECT DATA BY HAVING THE PROVIDER INSERT IT INTO THE DATABASE. LEGENDS FOR THE REST OF THE COLUMNS FOLLOW. IN ALL COLUMNS EXCEPT "KEYWORD UNIVERSE TESTED," BUBBLES THAT ARE MORE FILLED IN REPRESENT PROPERTIES THAT ARE BETTER FOR THE SCHEME AND WORSE FOR THE ATTACKER.

PRIOR KNOWLEDGE
- ●– CONTENTS OF FULL DATASET
- ◕– CONTENTS OF A SUBSET OF DATASET
- ◐– DISTRIBUTIONAL KNOWLEDGE OF DATASET
- ◔– DISTRIBUTIONAL KNOWLEDGE OF QUERIES
- ○– KEYWORD UNIVERSE

RUNTIME (IN # OF KEYWORDS)
- ●– MORE THAN QUADRATIC
- ◐– QUADRATIC
- ○– LINEAR

SENSITIVITY TO PRIOR KNOWLEDGE
- ●– HIGH
- ○– LOW
- ? – UNTESTED

KEYWORD UNIVERSE TESTED
- ●– > 1000
- ◐– 500 TO 1000
- ○– < 500

In summary, each protected search approach has a distinct leakage profile that results in qualitatively different attacks. If queries only touch a small portion of the dataset or the adversary only has a snapshot, the impact of leakage from `Custom` systems is less than from `Legacy` schemes. If queries regularly return a large fraction of the dataset, this distinction disappears and an `Obliv` scheme may be appropriate. Recently, Kellaris et al. [125] showed an attack on `Obliv` schemes, but it requires significantly smaller database and keyword universe sizes than attacks against non-`Obliv` schemes.

Open Problems: The area of leakage attacks against protected search is expanding. Published attacks consider attackers who insert specially crafted data records but have not considered an attacker who may issue crafted queries. Furthermore, all prior attacks have considered the leakage profile of the server. Future attacks should consider the implications of leakage to the querier and provider. Current attacks have targeted Equality and Range queries; we encourage the study of leakage attacks on other query types such as Boolean queries.

On the reverse side, it is important to understand what these leakage attacks mean in real-world application scenarios. Specifically, is it possible to identify appropriate real-world use-cases where the known leakage attacks do not disclose too much information? Understanding this will enable solutions that better span the security, performance, and functionality tradeoff space.

Lastly, on the defensive side we encourage designers to implement **Refresh** mechanisms. **Refresh** mechanisms have only been implemented for Equality systems.

IV. Extending Functionality

A. Query Composition

We now describe techniques to combine the base queries described in Section III (equality, Boolean, and range queries) to obtain richer queries. We restrict our attention to techniques that are black box (i.e., they do not depend on the implementation of the base queries).

As a general principle, schemes that support a given query type by composing base queries tend to have more leakage than schemes that natively support the same query type as a base query. However, using query composition, a scheme that supports the necessary base queries can be extended straightforwardly to support multiple query types, whereas supporting those all as base queries requires significant effort. Thus, we see value in advancing both base and composed queries.

Table IV summarizes the techniques we describe below. In the table and the text, we cite the first work proposing each approach, though we note that several ideas appear to have been developed independently and concurrently. We defer the description of string queries (substrings and wildcards) to Appendix A.

1) Equality using range: Equality queries can be supported using a range query scheme. To obtain the records equal to a, the querier performs a range query for the range $[a, a]$.

2) Disjunction of equalities/ranges using equality/range: Disjunctions of equalities or ranges can be supported using an equality or a range scheme, respectively. To obtain the records that equal any of a set of k keywords $w_1, \ldots, w_k$, the querier can perform an equality query for each keyword w_i and combine the results. Similarly, to retrieve all records that are in any of k ranges, the querier can perform a range query for each range and combine the results. This approach reveals to the server the leakage associated with each equality or range query, e.g., the exact or approximate number of records matching each clause (not just the number of records matching the disjunction overall).

3) Conjunction of equalities using equality: Conjunctions of equalities can be supported using an equality scheme. To supporting querying for records that match all of the keywords $w_1, \ldots, w_k$, one builds an equality scheme containing k-tuples of keywords. The querier then performs an equality search on the k-tuple representing her query to retrieve the records that contain all of those keywords. The storage for this approach grows exponentially with k but is viable for targeted keyword combinations or a small number of fields.

4) Stemming using equality: Stemming reduces words to their root form; stemming queries allow matching on word variations. For example, a stemming query for 'run' will also return results for 'ran' and 'running'. The Porter stemmer is a widely used algorithm [135], [136]. Stemming can be supported easily by using the stemmed version of keywords at both initialization and query time, and thus performing the match using a single equality query.

5) Proximity using equality: Proximity queries find values that are 'close' to the search term. Li et al. [137] support proximity queries by building an equality scheme associating each neighbor of any record with its set of neighbors in the dataset at initialization; a proximity query is then an equality query, which will return a record if it matches the queried value or is a neighbor of it. Boldyreva and Chenette [133] improve on the security of this scheme by revealing only pairwise neighbor relationships instead of neighbor sets. They also pad the number of inserted keywords to the maximum number of neighbors. This solution multiplies storage by the maximum number of neighbors of a record. If disjunctive searches are permitted, one can trade off storage space with the number of terms in the search.

Another approach uses locality-sensitive hashing [138], [139], which preserves closeness by mapping 'close' inputs to identical values and 'far' inputs to uncorrelated values. Proximity queries can be supported by inserting the output of a locality-sensitive hash as a keyword in an equality scheme. Returning only 'close' records requires matching the output of multiple hashes. Parameters vary widely depending on the notion of closeness. This approach has been demonstrated for Jaccard distance [140] and Hamming distance [137], [141]–[144].

6) Small-domain range query using equality [134]: To support range queries on a searchable attribute A with domain D, we build two equality-searchable indices. The first index

Composed Query	Base Query Calls	Additional Storage	Leakage	Work
1. Equality (EQ)	1 range	none	Same as range	—
2. Disjunction (OR) of k EQs (or ranges)	k EQs (or ranges)	none	Identifiers of records matching each clause, if EQ leaks $\geq$ ◔	—
3. Conjunction (AND) of k EQs	1 EQ	$\binom{\beta}{k}$	Same as EQ	—
4. Stemming	1 EQ	1	Identifiers of records sharing stem, if EQ leaks $\geq$ ◔	—
5. Proximity	1 EQ	ℓ	Identifiers of neighbor pairs, if EQ leaks $\geq$ ◔	[133]
6. Range w/ small domain	$(2+r)$ EQs	1	No leakage if refresh between queries	[134]
7. Range	OR of $(2\log m)$ EQs	$\log m$	Distributional info, if EQ leaks $\geq$ ◔	[16]
8. Negation	AND of 2 ranges	1	Same as OR of ranges	[16]
9. Substring ($\rho = \kappa$)	1 EQ	$\alpha - \kappa + 1$	Identifiers of records sharing κ-grams, if EQ leaks $\geq$ ◔	[22]
10. Substring ($\rho \leq \kappa$)	1 range	$\alpha - \kappa + 1$	Same as range, on κ-grams	[22]
11. Anchored Substring ($\rho \geq \kappa$)	AND of $(\rho - \kappa + 1)$ EQs	$\alpha - \kappa + 1$	If EQ leaks $\geq$ ◔, rec. ids. w/ κ-grams in same positions; if AND leaks # clauses, ρ	[18]
12. Substring	OR of $(\alpha - \kappa + 1)$ ANDs of $(\rho - \kappa + 1)$ EQs	$\alpha - \kappa + 1$	If EQ leaks $\geq$ ◔, rec. ids. w/ κ-grams in same positions; if AND leaks # clauses, ρ	[18]
13. Anchored Wildcard	AND of $(\rho - \kappa + 1)$ EQs	$\alpha - \kappa + 1$	If EQ leaks $\geq$ ◔, rec. ids. w/ κ-grams in same positions; if AND leaks # clauses, ρ	[18]
14. Wildcard	OR of $(\alpha - \kappa + 1)$ ANDs of $(\rho - \kappa + 1)$ EQs	$\alpha - \kappa + 1$	If EQ leaks $\geq$ ◔, rec. ids. w/ κ-grams in same positions; if AND leaks # clauses, ρ	[18]

TABLE IV

SUMMARY OF QUERY COMBINERS USING EQUALITY (EQ), CONJUNCTION (AND), DISJUNCTION (OR), AND RANGE BASE QUERY TYPES. STORAGE IS GIVEN AS ADDITIONAL STORAGE BEYOND THAT REQUIRED FOR THE BASE EQUALITY OR RANGE QUERIES, AS A MULTIPLICATIVE FACTOR OVER THE BASE STORAGE. COMPOSED QUERY LEAKAGE DEPENDS ON THE LEAKAGE OF THE BASE QUERIES USED; THE TABLE GIVES THE COMPOSED QUERY LEAKAGE IF THE BASE EQUALITY SCHEME LEAKS IDENTITIES. "ANCHORED" REFERS TO A SEARCH THAT OCCURS AT EITHER THE BEGINNING OR THE END OF A STRING.

BOOLEAN NOTATION
k = # OF CLAUSES IN BOOLEAN
β = MAX # OF KEYWORDS PER RECORD

PROXIMITY, RANGE NOTATION
ℓ = MAX # OF NEIGHBORS OF A RECORD
m = SIZE OF DOMAIN
r = # QUERY RESULTS

STRING NOTATION
κ = LENGTH OF GRAMS
ρ = LENGTH OF QUERY STRING
α = MAX LENGTH OF DATA STRING (PADDED IF NECESSARY)

maps each value $a \in D$ to the number of records in the database smaller than a and the number of records larger than a. With two equality queries into this index, the querier can learn the location of the lower and upper bounds of a range query. The second index is an ordered list of records sorted by A, from which the client reads the relevant subset.

This approach requires blinding factors to prevent the client from learning the positions of the results while still being able to search the second index [134]. Also, this approach only works for attributes with small domain, since the first index has size proportional to the domain size.

7) Large-domain range using equality and disjunction [16], [134]: Range queries can be performed over exponential size domains via range covers, which are a specialization of set covers that effectively pre-compute the results of canonical range queries that would be asked during a binary search of each record. For instance, consider the domain $D = [0, 8)$ with size $m = 8$. To insert a record with attribute $A = 3$, we insert keywords corresponding to each of the canonical ranges $[0, 8)$, $[0, 4)$, $[2, 4)$, and $[3, 4)$. Range queries are split into canonical ranges; for instance, the range $[2, 5)$ would be split into $[2, 4)$ and $[4, 5)$. Combining this technique with disjunctions yields range queries [16].

Demertzis et al. [145] provide a variety of range cover schemes with different tradeoffs between leakage, storage, and computation. At the extremes, they can support constant storage with query cost linear in the range size, or m^2 multiplicative storage with constant-sized keyword queries. They recommend a balanced approach similar to [16], [134], although their recommended scheme has false positives.

8) Negations using range and disjunction [16]: As above consider an ordered domain D with minimum and maximum values a_{min} and a_{max}, respectively. To search for all records not matching $A = a$, compute a disjunction of the queries $[a_{min}, a)$ and $(a, a_{max}]$.

B. The Functionality Gap

We now review gaps in query functionality based on current protected base and combined queries. Our discussion is divided among the three query bases from Section II-A.

a) Relational Algebra: Cartesian product, which corresponds to the JOIN keyword in SQL, has been demonstrated in `Legacy` schemes. The one `Custom` scheme that supports Cartesian product is the work of Kamara and Moataz [102], but their scheme does not support updates.

The JOIN keyword makes a system *relational*. Secure JOIN is a crucial capability for protected search systems. The key challenge is to create a data structure capable of linking different values that reveals no information to any party. This challenge also arises in Boolean `Custom` systems. Systems overcome this challenge by placing values that could be linked in a single joint data structure. It is difficult to scale this approach to the JOIN operation as the columns involved are not known ahead of time (and there are many more possibilities).

Open Problem: Support secure Cartesian product using `Custom` and `Obliv` approaches.

b) Associative Arrays: The main workhorse of associative arrays is the ability to quickly add and multiply arrays. `Legacy` schemes have shown how to support limited addition through the use of somewhat homomorphic encryption. There is extensive work on private addition and multiplication using secure computation. However, this problem has not received substantial attention in the protected search literature. We see adaptation of (parallelizable) arithmetic techniques into protected search as a key to supporting associative arrays.

Open Problem: Incorporate secure computation into protected search systems to support array $(+, \times)$.

In addition, associative arrays are often constructed for string objects. In this setting, multiplication and addition are usually replaced with the concatenate function and an application-defined 'minimum' function that selects one of the two values. Finding the minimum is connected to the comparison operation. The comparison operation has been identified as a core gadget in the secure computation literature [146], [147]. We encourage adaptation of this gadget to protected search.

Open Problem: Support protected queries to output the minimum of two strings.

c) Linear Algebra: The main gap in supporting linear algebra is how to privately multiply two matrices. This problem is made especially challenging as for different data types the addition $(+)$ and multiplication $(\times)$ operations may be defined arbitrarily. Furthermore, linear algebra databases store data as sparse matrices. Access patterns to a sparse matrix may leak about the contents. This problem has begun to receive attention in the learning literature [148] as matrix multiplication enables many linear classification approaches. However, current work requires specializing storage to a particular algorithm, such as shortest path [116], [149].

Open Problem: Support efficient secure matrix multiplication and storage.

V. From Queries to Database Systems

In addition to search, a DBMS enforces rules, defines data structures, and provides transactional guarantees to an application. In this section, we highlight important components that are affected by security and need to be addressed to enable a protected search system to become a full DBMS. We then discuss current protected search systems and their applicability for different DB settings.

A. Controls, Rules and Enforcement

Classical database security includes a broad set of control measures, including access control, inference control, flow control, and data encryption [150].

Access control assigns a principal such as a user, role, account, or program privileges to interact with objects like tables, records, columns, views, or operations in a given context [151]. *Discretionary* access control balances usability with security and is used in most applications. *Mandatory* access control is used where a strict hierarchy is important and available for individuals and data. *Inference control* is used with statistical databases and restricts the ability of a principal to infer a fact about a stored datum from the result returned by an aggregate function such as average or count. *Flow control* ensures that information in an object does not flow to another object of lesser privilege. *Data encryption* in classical systems is used for transmitting data from the database back to the client and user. Some systems also encrypt the data at rest and use fine-grained encryption for access control [152]. These techniques are covered in most database textbooks.

A new complementary approach is called *query control* [153]. Query control limits which queries are acceptable, not which objects are visible by a user. As an example, a user may be required to specify at least five columns in a query, ensuring the query is sufficiently "targeted." It enables database designers to match legal requirements written in this style. Query control can be expressed using a *query policy*, which regulates the set of query controls.

Most current protected search designs do not consider either an **authorizer** or **enforcer**. Integrating this functionality is an important part of maturing protected search and complements the cryptographic protections provided by the basic protocols.

B. Performance Characterization

Database system adoption depends on response time on the expected set of queries. Databases are highly tuned, often creating indices on the fly in response to queries. This makes fair and fast evaluation difficult. To address this challenge, we developed a performance evaluation platform. Our platform has been open-sourced with a BSD license (https://github.com/mitll-csa/). Design details can be found in [154]–[156]. It has been used to test protected search systems at scales of 10TB. Prior works [16], [17], [19], [22] report performance numbers generated by our platform. While the platform has been used to evaluate SQL-style databases it was designed with reusability and extensibility in mind to allow generalization to other types of databases.

Our platform evaluates: 1) integrity of responses and modifications (when occurring individually and while other operations are being performed) and 2) query latency and throughput under a wide variety of conditions. The system can vary environmental characteristics, the size of the database, query types, how many records will be returned by each query, and query policy. Each of these factors can be measured independently to create performance cross-sections.

In our experiments, we found protected search response time depends heavily on:

1) Network capacity, load, and number of records returned by a query. Protected search systems often have more rounds of communication and network traffic than unprotected databases.
2) The ordering of terms and subclauses within a query. Query planning is difficult for protected search systems as they do not know statistics of the data. Protected search generates a plan based on only query type.
3) The existence and complexity of rules (query policy and access control). Protected search systems use advanced

cryptography like multi-party computation to evaluate these rules, resulting in substantial overhead.

C. User Perceptions of Performance

We conducted a human-subjects usability evaluation to further the understanding of current protected search usability. This evaluation considered the performance of multiple protected search technologies and the perception of performance by human subjects (our procedure was approved by our Institutional Review Board). In this evaluation, subjects interacted with different protected search systems through an identical web interface. Here, we focus on thoughts shared by participants during discussion. (An informal overview of our procedure is in Appendix B.)

Our participants discussed several themes that are salient for furthering the usability of protected search:

- Participants cared more about predictability of response times than minimizing the mean response time. When response times were unpredictable, participants were unsure whether they should wait for a query to complete or do something else.
- Participants felt the protected technologies were slower than an unprotected system. Participants felt this performance was acceptable if it gave them access to additional data, but did not want to migrate current databases to a protected system. Note that this feedback is from end users, not administrators.
- Participants expected performance to be correlated with the number of records returned and the length of the query. Participants were surprised that different types of queries might have different performance characteristics.

D. Current Protected Search Databases

Some protected search systems have made the transition to full database solutions. These systems report performance analysis, perform rule enforcement, and support dynamic data.

These systems are summarized in Table V. CryptDB replicates most DBMS functionality with a performance overhead of under 30% [15]. This approach has been extended to NoSQL key-value stores [157], [158]. Arx is built on a NoSQL key-value store called mongoDB [63]. Arx reports a performance overhead of approximately 10% when used to replace the database of a web application (ShareLatex). Blind Seer [16] reports slowdown of between 20% and 300% for most queries, while OSPIR-OXT [18] report they occasionally outperform a baseline MySQL 5.5 system with a cold cache and are an order of magnitude slower than MySQL with a warm cache. The SisoSPIR system [22] reports performance slowdown of 500% compared to a baseline MySQL system on keyword equality and range queries.

Given these performance numbers, we now ask which solution, if any, is appropriate for different database settings.

1) Relational Algebra without Cartesian product: CryptDB, Blind Seer, OSPIR-OXT, and SisoSPIR all provide functionality that supports most of relational algebra except for the Cartesian product operation. These systems offer different performance/leakage tradeoffs. CryptDB is the fastest and easiest to deploy. However, once a column is used in a query, CryptDB reveals statistics about the entire dataset's value on this column. The security impact of this leakage should be evaluated for each application (see Section III-B). Blind Seer and OSPIR-OXT also leak information to the server but primarily on data returned by the query. Thus, they are appropriate in settings where a small fraction of the database is queried. Finally, SisoSPIR is appropriate if a large fraction of the data is regularly queried. However, SisoSPIR does not support Boolean queries, which is limiting in practice.

2) Full Relational Algebra: CryptDB is the only system for relational algebra that supports Cartesian product. (As stated, while Kamara and Moataz [102] support Cartesian product, but do not support dynamic data.)

3) Associative Array - NoSQL Key-Value: The Arx system built on mongoDB provides functionality necessary to support associative arrays. In addition, other commercial systems (e.g., Google's Encrypted BigQuery [29]) and academic works [157], [158] apply `Legacy` techniques to build a NoSQL protected system.

Blind Seer, OSPIR-OXT, and SisoSPIR have sufficient query functionality to support associative arrays. However, their techniques concentrate on query performance. Associative array databases often have insert rates of over a million records per second. The insert rates of Blind Seer, OSPIR-OXT, and SisoSPIR are multiple orders of magnitude smaller. Suppose a record is being updated. In an unprotected system this causes a small change to the primary index structure. However in the protected setting, if only a few locations are modified the server may learn about the statistics of the updated record. This creates a tension between efficiency and security. Efficient updates are even more difficult if the provider does not have the full unprotected dataset.

Open Problem: Construct `Custom` and `Obliv` techniques that can handle millions of inserts per second.

To support very large insert rates, NoSQL key-value stores commonly distribute the data across many machines. This introduces the challenge of synchronizing queries, updates, and data across these machines. This synchronization is difficult as none of the servers are supposed to know what queries, updates, or data they are processing!

Open Problem: Construct protected search systems that leverage distributed server architectures.

4) Linear Algebra and Others: No current protected search system supports the linear algebra basis used to implement complex graph databases. In addition, as federated and polystore databases emerge it will be important to interface between different protected search systems that are designed for different query bases.

Inherent Limitations: Protected search systems are still in development, so it is important to distinguish between transient limitations and inherent limitations of protected search. Protected search inherently reduces data visibility in order to prevent abuse. To achieve high performance under these

System	Equality	Boolean	Keyword	Range	Substring	Wildcard	Sum	Join	Update	Approach	# of parties	Code available	Multi-client	User auth.	Access control	Query policy	Leakage	Performance
	Supported Operations									Properties			Features					
CryptDB [15]	●	●	○	●	○	○	●	●	●	`Legacy`	2	●	●	●	●	○	◕	●
Arx [14]	●	○	○	●	○	○	●	●	●	`Custom`	2	○	○	○	○	○	◐	◐
BLIND SEER [16], [17]	●	●	●	●	○	○	○	○	●	`Custom`	3	○	●	○	○	●	◐	◐
OSPIR-OXT [18]–[21], [103], [104]	●	●	●	●	●	●	○	○	●	`Custom`	3	○	●	○	○	●	◐	◕
SisoSPIR [22]	●	○	●	●	●	○	○	○	○	`Obliv`	3	○	○	○	○	●	○	◐

TABLE V

THIS TABLE SUMMARIZES PROTECTED SEARCH DATABASES THAT HAVE BEEN DEVELOPED AND EVALUATED AT SCALE. THE *Supported Operations* COLUMNS DESCRIBE THE QUERIES NATURALLY SUPPORTED BY EACH SCHEME. *Properties* AND *Features* COLUMNS DESCRIBE THE SYSTEM AND AVAILABLE FUNCTIONALITY. FINALLY *Leakage* AND *Performance* DESCRIBE THE WHOLE, COMPLEX SYSTEM, AND ARE THEREFORE RELATIVE (VS. THE MORE PRECISELY DEFINED VALUES FOR INDIVIDUAL OPERATIONS USED EARLIER).

conditions, many design decisions such as the schema and the choice of which indices to build must be made before data is ingested and stored on the server. In particular, if an index has not been built for a particular field, then it simply cannot be searched without returning the entire database to the querier. In general, it is not possible to dynamically permit a type of search without retrieving the entire dataset.

Additionally, if the database malfunctions, debugging efforts are complicated by the reduced visibility into server processes and logs. More generally, protected search systems are more complicated to manage and don't yet have an existing community of qualified, certified administrators.

Throughout this work we've identified a few transient limitations that can (and should!) be mitigated with future advances. Each potential user must make her own judgment as to whether the value of improved security outweighs the performance limitations.

VI. CONCLUSION AND OUTLOOK

Several established and startup companies have commercialized protected search. Most of these products today use the `Legacy` technique, but we believe both `Custom` and `Obliv` approaches will find their way into products with broad user bases.

Governments and companies are finding value in *lacking* access to individuals' data [159]. Proactively protecting data mitigates the (ever-increasing) risk of server compromise, reduces the insider threat, can be marketed as a feature, and frees developers' time to work on other aspects of products and services. The recent HITECH US Health Care Law [160] establishes a requirement to disclose breaches involving more than 500 patients but exempts companies if the data is encrypted: "if your practice has a breach of encrypted data [...] it would not be considered a breach of unsecured data, and you would not have to report it" [161].

Protected database technology can also open up new markets, such as those cases where there is great value in recording and sharing information but the risk of data spills is too high For example, companies recognize the value of sharing cyber threat and attack information [162], but uncontrolled sharing of this information presents a risk to reputation and intellectual property.

This paper provides a snapshot of current protected search solutions. There is currently no dominant solution for all use cases. Adopters need to understand system characteristics and tradeoffs for their use case.

Protected databases will see widespread adoption. Protected search has developed rapidly since 2000, advancing from linear time equality queries on static data to complex searches on dynamic data, now within overhead between 30%-500% over standard SQL.

At the same time, the database landscape is rapidly changing, specializing, adding new functionality, and federating approaches. Integrating protected search in a unified design requires close interaction between cryptographers, protected search designers, and database experts. To spur that integration, we describe a three pronged approach to this collaboration: 1) developing base queries that are useful in many applications, 2) understanding how to combine queries to support multiple applications, and 3) rapidly applying techniques to emerging database technologies.

DBMSs are more than just efficient search systems; they are highly optimized and complex systems. Protected search has shown that database and cryptography communities can work together. The next step is to transform protected search systems into protected DBMSs.

ACKNOWLEDGMENTS

The authors thank David Cash, Carl Landwehr, Konrad Vesey, Charles Wright, and the anonymous reviewers for helpful feedback in improving this work.

REFERENCES

[1] R. Powers and D. Beede, "Fostering innovation, creating jobs, driving better decisions: The value of government data," Office of the Chief Economist, Economics and Statistics Administration, US Department of Commerce, July 2014.

[2] G. S. Linoff and M. J. Berry, *Mining the Web: Transforming Customer Data into Customer Value*. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[3] "Big & fast data: The rise of insight-driven business," 2015. [Online]. Available: https://www.capgemini.com/resource-file-access/resource/pdf/big_fast_data_the_rise_of_insight-driven_business-report.pdf

[4] B. Mons, H. van Haagen, C. Chichester, P.-B. t. Hoen, J. T. den Dunnen, G. van Ommen, E. van Mulligen, B. Singh, R. Hooft, M. Roos, J. Hammond, B. Kiesel, B. Giardine, J. Velterop, P. Groth, and E. Schultes, "The value of data," *Nat Genet*, vol. 43, no. 4, pp. 281–283, Apr 2011. [Online]. Available: http://dx.doi.org/10.1038/ng0411-281

[5] Mandiant, http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf, Feb 2013.

[6] M. Motoyama, D. McCoy, K. Levchenko, S. Savage, and G. M. Voelker, "An analysis of underground forums," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 71–80.

[7] N. Y. Times, "Hacking linked to China exposes millions of U.S. workers," http://www.nytimes.com/2015/06/05/us/breach-in-a-federal-computer-system-exposes-personnel-data.html, June 4, 2015, accessed: 2015-07-09.

[8] ——, "9 recent cyberattacks against big businesses," http://www.nytimes.com/interactive/2015/02/05/technology/recent-cyberattacks.html, February 5, 2015, accessed: 2015-07-09.

[9] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2000, pp. 44–55.

[10] O. Pandey and Y. Rouselakis, "Property preserving symmetric encryption," in *EUROCRYPT 2012*, ser. LNCS, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer, Heidelberg, Apr. 2012, pp. 375–391.

[11] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *ACM CCS 06*, A. Juels, R. N. Wright, and S. Vimercati, Eds. ACM Press, Oct. / Nov. 2006, pp. 79–88.

[12] B. Chor, N. Gilboa, and M. Naor, "Private information retrieval by keywords," Cryptology ePrint Archive, Report 1998/003, 1998, http://eprint.iacr.org/1998/003.

[13] O. Goldreich, "Towards a theory of software protection and simulation by oblivious RAMs," in *19th ACM STOC*, A. Aho, Ed. ACM Press, May 1987, pp. 182–194.

[14] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system," Cryptology ePrint Archive, Report 2016/591, 2016, http://eprint.iacr.org/2016/591.

[15] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: processing queries on an encrypted database," *Commun. ACM*, vol. 55, no. 9, pp. 103–111, 2012. [Online]. Available: http://doi.acm.org/10.1145/2330667.2330691

[16] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. D. Keromytis, and S. Bellovin, "Blind seer: A scalable private DBMS," in *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 359–374.

[17] B. A. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, and S. M. Bellovin, "Malicious-client security in Blind Seer: A scalable private DBMS," in *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 395–410.

[18] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *CRYPTO 2013, Part I*, ser. LNCS, R. Canetti and J. A. Garay, Eds., vol. 8042. Springer, Heidelberg, Aug. 2013, pp. 353–373.

[19] S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *ACM CCS 13*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM Press, Nov. 2013, pp. 875–888.

[20] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *NDSS 2014*. The Internet Society, Feb. 2014.

[21] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M.-C. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *ESORICS 2015, Part II*, ser. LNCS, G. Pernul, P. Y. A. Ryan, and E. R. Weippl, Eds., vol. 9327. Springer, Heidelberg, Sep. 2015, pp. 123–145.

[22] Y. Ishai, E. Kushilevitz, S. Lu, and R. Ostrovsky, "Private large-scale databases with distributed searchable symmetric encryption," in *CT-RSA 2016*, ser. LNCS, K. Sako, Ed., vol. 9610. Springer, Heidelberg, Feb. / Mar. 2016, pp. 90–107.

[23] "Bitglass." [Online]. Available: http://www.bitglass.com/

[24] "Ciphercloud." [Online]. Available: http://www.ciphercloud.com

[25] "Cipherquery." [Online]. Available: https://privatemachines.com

[26] "Crypteron." [Online]. Available: https://www.crypteron.com/

[27] "IQrypt." [Online]. Available: http://iqrypt.com/

[28] "Kryptnostic." [Online]. Available: https://www.kryptnostic.com/

[29] Google, "Encrypted BigQuery client." [Online]. Available: https://github.com/google/encrypted-bigquery-client

[30] Microsoft Corporation, "Always Encrypted (Database Engine) - SQL Server 2016." [Online]. Available: https://msdn.microsoft.com/en-us/library/mt163865.aspx

[31] ——, "Always Encrypted Cryptography - SQL Server 2016." [Online]. Available: https://msdn.microsoft.com/en-us/library/mt653971.aspx

[32] "PreVeil." [Online]. Available: https://www.preveil.com/

[33] "Skyhigh networks: Cloud security software." [Online]. Available: https://www.skyhighnetworks.com

[34] "StealthMine." [Online]. Available: http://stealthmine.com/

[35] "ZeroDB." [Online]. Available: https://zerodb.com/

[36] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[37] M. Stonebraker and U. Cetintemel, "One size fits all: an idea whose time has come and gone," in *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 2005, pp. 2–11.

[38] J. D. Ullman, *A first course in database systems*. Pearson Education India, 1982.

[39] M. Stonebraker and J. M. Hellerstein, *Readings in database systems*. Morgan Kaufmann Publishers, 1988.

[40] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys (CSUR)*, vol. 15, no. 4, pp. 287–317, 1983.

[41] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[42] A. Pavlo and M. Aslett, "What's really new with NewSQL?" *SIGMOD Record*, 2016.

[43] A. Elmore, J. Duggan, M. Stonebraker, M. Balazinska, U. Cetintemel, V. Gadepally, J. Heer, B. Howe, J. Kepner, T. Kraska *et al.*, "A demonstration of the BigDAWG polystore system," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1908–1911, 2015.

[44] V. Gadepally, P. Chen, J. Duggan, A. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, and M. Stonebraker, "The BigDAWG polystore system and architecture," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2016, pp. 1–6.

[45] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker *et al.*, "Demonstration of the Myria big data management service," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 881–884.

[46] R. A. Van De Geijn and E. S. Quintana-Ortí, *The science of programming matrix computations*, 2008.

[47] J. Kepner and V. Gadepally, "Adjacency matrices, incidence matrices, database schemas, and associative arrays," in *International Parallel & Distributed Processing Symposium Workshops (IPDPSW). IEEE*, 2014.

[48] V. Gadepally, J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, L. Edwards, M. Hubbell, P. Michaleas, J. Mullen *et al.*, "D4M: Bringing associative arrays to database engines," in *High Performance Extreme Computing Conference (HPEC), 2015 IEEE*. IEEE, 2015, pp. 1–6.

[49] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 1099–1110.

[50] J. Kepner, V. Gadepally, D. Hutchison, H. Jananthan, T. Mattson, S. Samsi, and A. Reuther, "Associative array model of SQL, NoSQL, and NewSQL databases," in *2016 IEEE High Performance Extreme Computing Conference*, 2016.

[51] D. J. Abadi, "Data management in the cloud: limitations and opportunities." *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 3–12, 2009.

[52] "MySQL." [Online]. Available: https://www.mysql.com/

[53] K. Loney, *Oracle database 10g: the complete reference*. McGraw-Hill/Osborne, 2004.

[54] M. Stonebraker and L. A. Rowe, *The design of Postgres*. ACM, 1986, vol. 15, no. 2.

[55] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, p. 8, 2013.

[56] N. Shamgunov, "The MemSQL in-memory database system." in *IMDM@ VLDB*, 2014.

[57] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark SQL: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.

[58] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic *et al.*, "Towards heterogeneous multimedia information systems: The garlic approach," in *Research Issues in Data Engineering, 1995: Distributed Object Management, Proceedings. RIDE-DOM'95. Fifth International Workshop on*. IEEE, 1995, pp. 124–131.

[59] "IBM DB2." [Online]. Available: http://www.ibm.com/analytics/us/en/technology/db2/

[60] D. Pritchett, "BASE: An ACID alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.

[61] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout *et al.*, "Achieving 100,000,000 database inserts per second using accumulo and D4M," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2014, pp. 1–6.

[62] L. George, *HBase: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[63] "mongoDB." [Online]. Available: https://www.mongodb.com/

[64] J. Webber, "A programmatic introduction to Neo4j," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. ACM, 2012, pp. 217–218.

[65] "IBM system G." [Online]. Available: http://systemg.research.ibm.com/

[66] P. G. Brown, "Overview of sciDB: large scale array storage, processing and analysis," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 963–968.

[67] N. Li, *Scalable database query processing*. Johns Hopkins University, 2012.

[68] J. M. Smith and P. Y.-T. Chang, "Optimizing the performance of a relational algebra database interface," *Communications of the ACM*, vol. 18, no. 10, pp. 568–579, 1975.

[69] J. Kepner, D. Bader, A. Buluç, J. Gilbert, T. Mattson, and H. Meyerhenke, "Graphs, matrices, and the GraphBLAS: Seven good reasons," *Procedia Computer Science*, vol. 51, pp. 2453–2462, 2015.

[70] V. Gadepally, J. Bolewski, D. Hook, D. Hutchison, B. Miller, and J. Kepner, "Graphulo: Linear algebra graph kernels for NoSQL databases," in *International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2015.

[71] D. Hutchison, J. Kepner, V. Gadepally, and A. Fuchs, "Graphulo implementation of server-side sparse matrix multiply in the accumulo database," in *High Performance Extreme Computing Conference (HPEC), 2015 IEEE*. IEEE, 2015, pp. 1–7.

[72] Microsoft Corporation, "Database-level roles." [Online]. Available: https://msdn.microsoft.com/en-us/library/ms189121.aspx

[73] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 18:1–18:51, August 2014. [Online]. Available: http://doi.acm.org/10.1145/2636328

[74] P. Grubbs, R. McPherson, M. Naveed, T. Ristenpart, and V. Shmatikov, "Breaking web applications built on top of encrypted data," in *ACM CCS 16*. ACM Press, 2016, pp. 1353–1364.

[75] S. Kamara, "Structured encryption and leakage suppression," presented at Encryption for Secure Search and Other Algorithms, Bertinoro, Italy, June 2015.

[76] S. Bajaj and R. Sion, "TrustedDB: A trusted hardware-based database with privacy and data confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752–765, 2014.

[77] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164.

[78] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *20th ACM STOC*. ACM Press, May 1988, pp. 1–10.

[79] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *19th ACM STOC*, A. Aho, Ed. ACM Press, May 1987, pp. 218–229.

[80] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *41st ACM STOC*, M. Mitzenmacher, Ed. ACM Press, May / Jun. 2009, pp. 169–178.

[81] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 309–325.

[82] C. Gentry, S. Halevi, and N. P. Smart, "Better bootstrapping in fully homomorphic encryption," in *PKC 2012*, ser. LNCS, M. Fischlin, J. Buchmann, and M. Manulis, Eds., vol. 7293. Springer, Heidelberg, May 2012, pp. 1–16.

[83] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *54th FOCS*. IEEE Computer Society Press, Oct. 2013, pp. 40–49.

[84] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *36th FOCS*. IEEE Computer Society Press, Oct. 1995, pp. 41–50.

[85] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin, "Protecting data privacy in private information retrieval schemes," in *30th ACM STOC*. ACM Press, May 1998, pp. 151–160.

[86] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated data structures for graph and geometric searching," in *CT-RSA 2003*, ser. LNCS, M. Joye, Ed., vol. 2612. Springer, Heidelberg, Apr. 2003, pp. 295–313.

[87] C. Papamanthou and R. Tamassia, "Time and space efficient algorithms for two-party authenticated data structures," in *ICICS 07*, ser. LNCS, S. Qing, H. Imai, and G. Wang, Eds., vol. 4861. Springer, Heidelberg, Dec. 2008, pp. 1–15.

[88] M. Etemad and A. Küpçü, "Database outsourcing with hierarchical authenticated data structures," in *ICISC 13*, ser. LNCS, H.-S. Lee and D.-G. Han, Eds., vol. 8565. Springer, Heidelberg, Nov. 2014, pp. 381–399.

[89] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, "ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data," in *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 271–286.

[90] J. H. Ahn, D. Boneh, J. Camenisch, S. Hohenberger, a. shelat, and B. Waters, "Computing on authenticated data," in *TCC 2012*, ser. LNCS, R. Cramer, Ed., vol. 7194. Springer, Heidelberg, Mar. 2012, pp. 1–20.

[91] A. Hamlin, N. Schear, E. Shen, M. Varia, S. Yakoubov, and A. Yerukhimovich, "Cryptography for big data security," in *Big Data: Storage, Sharing, and Security*, F. Hu, Ed. Taylor & Francis LLC, CRC Press, 2016.

[92] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *CRYPTO 2007*, ser. LNCS, A. Menezes, Ed., vol. 4622. Springer, Heidelberg, Aug. 2007, pp. 535–552.

[93] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004, pp. 563–574. [Online]. Available: http://doi.acm.org/10.1145/1007568.1007632

[94] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *EUROCRYPT 2009*, ser. LNCS, A. Joux, Ed., vol. 5479. Springer, Heidelberg, Apr. 2009, pp. 224–241.

[95] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *CRYPTO 2011*, ser. LNCS, P. Rogaway, Ed., vol. 6841. Springer, Heidelberg, Aug. 2011, pp. 578–595.

[96] C. Mavroforakis, N. Chenette, A. O'Neill, G. Kollios, and R. Canetti, "Modular order-preserving encryption, revisited," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 763–777. [Online]. Available: http://doi.acm.org/10.1145/2723372.2749455

[97] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 463–477.

[98] P. Grofig, M. Härterich, I. Hang, F. Kerschbaum, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert, "Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data," in *Sicherheit*, 2014, pp. 115–125. [Online]. Available: http://subs.emis.de/LNI/Proceedings/Proceedings228/article7.html

[99] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *ASIACRYPT 2010*, ser. LNCS, M. Abe, Ed., vol. 6477. Springer, Heidelberg, Dec. 2010, pp. 577–594.

[100] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 639–654.

[101] R. Bost, "$\Sigma o \phi o \varsigma$: Forward secure searchable encryption," in *ACM CCS 16*. ACM Press, 2016, pp. 1143–1154.

[102] S. Kamara and T. Moataz, "SQL on structurally-encrypted databases," Cryptology ePrint Archive, Report 2016/453, 2016, http://eprint.iacr.org/2016/453.

[103] ——, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *EUROCRYPT 2017*, 2017.

[104] T. Moataz, "Searchable symmetric encryption: Implementation of 2Lev, ZMF, IEX-2Lev, IEX-ZMF," https://github.com/orochi89/Clusion.

[105] D. Cash and S. Tessaro, "The locality of searchable symmetric encryption," in *EUROCRYPT 2014*, ser. LNCS, P. Q. Nguyen and E. Oswald, Eds., vol. 8441. Springer, Heidelberg, May 2014, pp. 351–368.

[106] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *FC 2013*, ser. LNCS, A.-R. Sadeghi, Ed., vol. 7859. Springer, Heidelberg, Apr. 2013, pp. 258–274.

[107] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *NDSS 2014*. The Internet Society, Feb. 2014.

[108] A. C.-C. Yao, "How to generate and exchange secrets (extended abstract)," in *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 162–167.

[109] M. Chase and E. Shen, "Substring-searchable symmetric encryption," *PoPETs*, vol. 2015, no. 2, pp. 263–281, 2015. [Online]. Available: http://www.degruyter.com/view/j/popets.2015.2015.issue-2/popets-2015-0014/popets-2015-0014.xml

[110] T. Boelter, R. Poddar, and R. A. Popa, "A secure one-roundtrip index for range queries," Cryptology ePrint Archive, Report 2016/568, 2016, http://eprint.iacr.org/2016/568.

[111] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich, "POPE: Partial order preserving encoding," in *ACM CCS 16*. ACM Press, 2016, pp. 1131–1142.

[112] F. Baldimtsi and O. Ohrimenko, "Sorting and searching behind the curtain," in *FC 2015*, ser. LNCS, R. Böhme and T. Okamoto, Eds., vol. 8975. Springer, Heidelberg, Jan. 2015, pp. 127–146.

[113] M. Strizhov and I. Ray, "Multi-keyword similarity search over encrypted cloud data," Cryptology ePrint Archive, Report 2015/137, 2015, http://eprint.iacr.org/2015/137.

[114] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *TCC 2009*, ser. LNCS, O. Reingold, Ed., vol. 5444. Springer, Heidelberg, Mar. 2009, pp. 457–473.

[115] C. Bösch, Q. Tang, P. H. Hartel, and W. Jonker, "Selective document retrieval from encrypted database," in *ISC 2012*, ser. LNCS, D. Gollmann and F. C. Freiling, Eds., vol. 7483. Springer, Heidelberg, Sep. 2012, pp. 224–241.

[116] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "GRECS: Graph encryption for approximate shortest distance queries," in *ACM CCS 15*, I. Ray, N. Li, and C. Kruegel:, Eds. ACM Press, Oct. 2015, pp. 504–517.

[117] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[118] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," in *ACM CCS 13*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM Press, Nov. 2013, pp. 299–310.

[119] M. Naveed, "The fallacy of composition of oblivious RAM and searchable encryption," Cryptology ePrint Archive, Report 2015/668, 2015, http://eprint.iacr.org/2015/668.

[120] D. S. Roche, A. J. Aviv, and S. G. Choi, "A practical oblivious map data structure with secure deletion and history independence," in *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2016, pp. 178–197.

[121] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption," ser. LNCS. Springer, Heidelberg, Aug. 2016, pp. 563–592.

[122] S. Lu and R. Ostrovsky, "How to garble RAM programs," in *EUROCRYPT 2013*, ser. LNCS, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Springer, Heidelberg, May 2013, pp. 719–734.

[123] T. Moataz and E.-O. Blass, "Oblivious substring search with updates," Cryptology ePrint Archive, Report 2015/722, 2015, http://eprint.iacr.org/2015/722.

[124] S. Faber, S. Jarecki, S. Kentros, and B. Wei, "Three-party ORAM for secure computation," in *ASIACRYPT 2015, Part I*, ser. LNCS, T. Iwata and J. H. Cheon, Eds., vol. 9452. Springer, Heidelberg, Nov. / Dec. 2015, pp. 360–385.

[125] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1329–1340. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978386

[126] E. Chen, I. Gomez, B. Saavedra, and J. Yucra, "Cocoon: Encrypted substring search," https://courses.csail.mit.edu/6.857/2016/files/29.pdf, May 2015, accessed: 2016-07-15.

[127] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, 2016, pp. 707–720. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhang

[128] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, 2015, pp. 668–679. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813700

[129] D. Pouliot and C. V. Wright, "The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, 2015, pp. 644–655. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813651

[130] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *23rd ACM Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, 2016.

[131] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-abuse attacks against order-revealing encryption," Cryptology ePrint Archive, Report 2016/895, http://eprint.iacr.org/2016/895.

[132] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[133] A. Boldyreva and N. Chenette, "Efficient fuzzy search on encrypted data," in *FSE 2014*, ser. LNCS, C. Cid and C. Rechberger, Eds., vol. 8540. Springer, Heidelberg, Mar. 2015, pp. 613–633.

[134] G. D. Crescenzo and A. Ghosh, "Privacy-preserving range queries from keyword queries," in *Data and Applications Security and Privacy XXIX*, ser. LNCS, vol. 9149. Springer, 2015, pp. 35–50.

[135] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[136] P. Willett, "The porter stemming algorithm: then and now," *Program*, vol. 40, no. 3, pp. 219–223, 2006.

[137] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 15-19 March 2010, San Diego, CA, USA*, 2010, pp. 441–445. [Online]. Available: http://dx.doi.org/10.1109/INFCOM.2010.5462196

[138] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[139] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, vol. 99, no. 6, 1999, pp. 518–529.

[140] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *IEEE 28th International Conference on Data Engineering (ICDE)*, 2012, pp. 1156–1167. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2012.23

[141] H. Park, B. H. Kim, D. H. Lee, Y. D. Chung, and J. Zhan, "Secure similarity search," in *2007 IEEE International Conference on Granular Computing, GrC 2007, San Jose, California, USA, 2-4 November 2007*, 2007, p. 598. [Online]. Available: http://dx.doi.org/10.1109/GRC.2007.140

[142] M. Adjedj, J. Bringer, H. Chabanne, and B. Kindarji, "Biometric identification over encrypted data made feasible," in *Information Systems Security, 5th International Conference, ICISS 2009, Kolkata, India, December 14-18, 2009, Proceedings*, 2009, pp. 86–100. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10772-6_8

[143] J. Bringer, H. Chabanne, and B. Kindarji, "Error-tolerant searchable encryption," in *Proceedings of IEEE International Conference on Communications, ICC 2009, Dresden, Germany, 14-18 June 2009*, 2009, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/ICC.2009.5199004

[144] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012*, 2012, pp. 451–459. [Online]. Available: http://dx.doi.org/10.1109/INFCOM.2012.6195784

[145] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *ACM SIGMOD/PODS Conference*, 2016.

[146] I. Damgard, M. Geisler, and M. Kroigard, "Homomorphic encryption and secure comparison," *International Journal of Applied Cryptography*, vol. 1, no. 1, pp. 22–31, 2008.

[147] F. Kerschbaum, D. Biswas, and S. de Hoogh, "Performance comparison of secure comparison protocols," in *Database and Expert Systems Application, 2009. DEXA'09. 20th International Workshop on*. IEEE, 2009, pp. 133–136.

[148] S. Han and W. K. Ng, "Privacy-preserving linear fisher discriminant analysis," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2008, pp. 136–147.

[149] X. S. Wang, K. Nayak, C. Liu, T.-H. H. Chan, E. Shi, E. Stefanov, and Y. Huang, "Oblivious data structures," in *ACM CCS 14*, G.-J. Ahn, M. Yung, and N. Li, Eds. ACM Press, Nov. 2014, pp. 215–226.

[150] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*. Boston, MA, USA: Addison-Wesley, 2011.

[151] E. Bertino and R. Sandhu, "Database security-Concepts, Approaches, and Challenges," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, 2005.

[152] A. Fuchs, "Accumulo–extensions to Google's Bigtable design," *National Security Agency, Tech. Rep*, 2012.

[153] IARPA, "Broad agency announcement IARPA-BAA-11-01: Security and privacy assurance research (SPAR) program." February 2011. [Online]. Available: https://www.fbo.gov/notices/c55e38dbde30cb668f687897d8f01e69

[154] A. Hamlin and J. Herzog, "A test-suite generator for database systems," in *2014 IEEE High Performance Extreme Computing Conference*, 2014, pp. 1–6.

[155] M. Varia, B. Price, N. Hwang, A. Hamlin, J. Herzog, J. Poland, M. Reschly, S. Yakoubov, and R. K. Cunningham, "Automated assessment of secure search systems," *Operating Systems Review*, vol. 49, no. 1, pp. 22–30, 2015.

[156] M. Varia, S. Yakoubov, and Y. Yang, "HEtest: A homomorphic encryption testing framework," in *FC 2015 Workshops*, ser. LNCS, M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds., vol. 8976. Springer, Heidelberg, Jan. 2015, pp. 213–230.

[157] J. Kepner, V. Gadepally, P. Michaleas, N. Schear, M. Varia, A. Yerukhimovich, and R. K. Cunningham, "Computing on masked data: a high performance method for improving big data veracity," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2014, pp. 1–6.

[158] V. Gadepally, B. Hancock, B. Kaiser, J. Kepner, P. Michaleas, M. Varia, and A. Yerukhimovich, "Computing on masked data to improve the security of big data," in *IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, 2015, pp. 1–6.

[159] B. Schneier, "Data is a toxic asset," 2016. [Online]. Available: https://www.schneier.com/essays/archives/2016/03/data_is_a_toxic_asse.html

[160] D. Blumenthal, "Launching HITECH," *New England Journal of Medicine*, vol. 362, no. 5, pp. 382–385, 2010.

[161] "The Office of the National Coordinator for Health Information Technology", "Guide to privacy and security of electronic health information," 2015. [Online]. Available: https://www.healthit.gov/sites/default/files/pdf/privacy/privacy-and-security-guide.pdf

[162] S. Barnum, "Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX)," *MITRE Corporation*, vol. 11, 2012.

[163] "Amazon Web Services (AWS) - cloud computing services." [Online]. Available: https://aws.amazon.com/

APPENDIX A
SUBSTRING AND WILDCARD QUERY COMBINERS

9) Bounded-length substring using keyword equality [22]: Searches for substrings of a fixed length κ can be supported simply by inserting all length-κ substrings (κ-grams) into an equality-searchable index during initialization. Given a field with maximum length α, this techniques requires adding $\alpha - \kappa$ keywords during insertion and making one keyword search during query execution.

10) Short substring using range [22]: By inserting the κ-grams into a range index, queries for substrings of length up to κ can also be supported. We explain by example: one can query for the two-character string "hi" by searching for the range $[hia, hiz]$ in an index of length-3 substrings.

11) Anchored substring using conjunction [18]: We now consider the converse of the above situation: supporting searches of long substrings of length *at least* κ, with storage overhead decreasing in κ. We begin with an "anchored" search, where the substring occurs either at the beginning or end of the string.

By way of example, suppose we wish to support substring searches on the record a ="teststring". In a conjunction-searchable index, we insert κ-grams of the string along with their location (1, "tes"), (2, "est"), ..., (8, "ing"). Now to search for all records containing "test" the client asks for all records matching both (1, "tes") and (2, "est"). Searching from the end of the string can be accomplished using negative indexing; using (-1, "ing"), (-2, "rin"), (-3, "tri"), ..., (-8, "tes") in the above example.

12) Substring using disjunction of conjunctions [18]: Removing the anchoring restriction from the above technique requires the use of disjunctions, since the starting location of the substring is unknown. To find the substring "test" the querier must search for a conjunction of (i, "tes") and (i+1, "est") for any starting position i. The number of terms in this formula depends on the maximum string length.

13) and 14) Wildcard using conjunctions [18]: The above technique also supports single-character wildcard queries. For instance, to search for "tes_str", the client asks for a conjunction of (1, "tes") and (5, "str"). Note that the κ-gram length of letters is required on either side of the wildcard. This technique can be extended for unanchored queries as above, and it supports multiple character wildcards by incrementing the expected positions of the κ-grams.

APPENDIX B
PROCEDURE FOR PILOT STUDY

We installed and configured multiple protected search systems. For each, we ingested ten million records of real application data, and conducted sessions with 10 users over a ten-day period. Our Institutional Review Board reviewed our protocols and questionnaires, determined that they represented a minimal risk, and approved the procedure. Software for the

procedure resided in an Amazon Web Services (AWS) [163] network. Data was drawn from a genuine application source and was converted to a single, static table with over one hundred columns and ten million records.

Participants had a mix of technical and non-technical backgrounds, with six men and four women. All participants had prior experience interacting with web interfaces that use database backends to present results. Participants were aware that they were using different systems but systems were identified only by a single letter. Participants were not given any information about the capabilities of the technologies.

Each participant took part in three types of sessions, each of which lasted 30 minutes: 1) training on the web interface; 2) scripted interaction with each of the technologies; and 3) exploratory sessions with each of the technologies. Users interacted with the secure database technology through a web application which included a visual query builder which queried the underlying secure database. Participants interacted with the visual query builder to create queries. Then, the web server submitted the query to the protected search system.