Towards Secure Data Outsourcing

Radu Sion

Network Security and Applied Cryptography Lab
Computer Science, Stony Brook University
sion@cs.stonybrook.edu

Abstract. The networked and increasingly ubiquitous nature of today's data management services mandates assurances to detect and deter malicious or faulty behavior. This is particularly relevant for outsourced data frameworks in which clients place data management with specialized service providers. Clients are reluctant to place sensitive data under the control of a foreign party without assurances of confidentiality. Additionally, once outsourced, privacy and data access correctness (data integrity and query completeness) become paramount. Today's solutions are fundamentally insecure and vulnerable to illicit behavior, because they do not handle these dimensions.

In this chapter we will explore the state of the art in data outsourcing mechanisms providing strong security assurances of (1) *correctness*, (2) *confidentiality*, and (3) data access *privacy*.

There exists a strong relationship between such assurances; for example, the lack of access pattern privacy usually allows for statistical attacks compromising data confidentiality. Confidentiality can be achieved by data encryption. However, to be practical, outsourced data services should allow expressive client queries (e.g., relational joins with arbitrary predicates) without compromising confidentiality. This is a hard problem because decryption keys cannot be directly provided to potentially untrusted servers. Moreover, if the remote server cannot be fully trusted, protocol correctness become essential.

Here we will discuss query mechanisms targeting outsourced relational data that (i) ensure queries have been executed with integrity and completeness over their respective target data sets, (ii) allow queries to be executed with confidentiality over encrypted data, (iii) guarantee the privacy of client queries and data access patterns. We will then propose protocols that adapt to the existence of *trusted hardware* — so critical functionality can be delegated securely from clients to servers. We have successfully started exploring the feasibility of such solutions for providing assurances for query execution and the handling of binary predicate JOINs with full privacy in outsourced scenarios.

The total cost of ownership of data management infrastructure is 5–10 times greater than the hardware costs, and more data is produced and lives digitally every day. In the coming years, secure, robust, and efficient outsourced data management will be demanded by users. It is thus important to finally achieve outsourced data management a trustworthy solution, viable in both personal-level and large corporate settings.

1 Introduction

Today, sensitive data is being managed on remote servers maintained by third party outsourcing vendors. This is because the total cost of data management is 5–10 times higher than the initial acquisition costs [61]. In such an outsourced "database as a service" [72] model, *clients* outsource data management to a "database service provider" that provides online access mechanisms for querying and managing the hosted data sets.

This is advantageous and significantly more affordable for parties with limited abilities to manage large in-house data centers of potentially large resource footprints. By comparison, database service providers [1–6, 6–9, 11–15] – ranging from corporate-level services such as the IBM Data Center Outsourcing Services to personal level database hosting – have the advantage of expertize consolidation. More-over, they are likely to be able to offer the service much cheaper, with increased service availability (e.g. uptime) guarantees.

Notwithstanding these clear advantages, a data outsourcing paradigm faces significant challenges to widespread adoption, especially in an online, untrusted environment. Current privacy guarantees of such services are at best declarative and often subject customers to unreasonable fine-print clauses—e.g., allowing the server operator (and thus malicious attackers gaining access to its systems) to use customer behavior and content for commercial, profiling, or governmental surveillance purposes [52]. Clients are naturally reluctant to place sensitive data under the control of a foreign party without strong security assurances of *correctness*, *confidentiality*, and data access *privacy*. These assurances are essential for data outsourcing to become a sound and truly viable alternative to in-house data management. However, developing assurance mechanisms in such frameworks is challenging because the data is placed under the authority of an external party whose honest behavior is not guaranteed but rather needs to be ensured by this very solution.

In this chapter, we will explore the challenges of designing and implementating robust, efficient, and scalable relational data outsourcing mechanisms, with strong security assurances of *correctness*, *confidentiality*, and data access *privacy*. This is important because today's outsourced data services are fundamentally insecure and vulnerable to illicit behavior, as they do not handle all three dimensions consistently and there exists a strong relationship between such assurances: e.g., the lack of access pattern privacy usually allows for statistical attacks compromising data confidentiality. Even if privacy and confidentiality are in place, to be practical, outsourced data services should allow sufficiently expressive client queries (e.g., relational operators such as JOINs with arbitrary predicates) without compromising confidentiality. This is a hard problem because in most cases decryption keys cannot be directly provided to potentially untrusted database servers. Moreover, result completeness and data integrity (i.e., correctness) become essential. Therefore, solutions that do not address these dimensions are incomplete and insecure.

We will explore designs for outsourced relational data query mechanisms that (i) ensure queries have been executed with *integrity and completeness* over their

respective target data sets, (ii) allow queries to be executed with *confidentiality* over encrypted data, and (iii) guarantee the *privacy* of client queries and data access patterns:

Correctness. Clients should be able to verify the integrity and completeness of any results the server returns. For example, when executing a JOIN query, they should be able to verify that the server returned *all* matching tuples.

Confidentiality. The data being stored on the server should not be decipherable either during transit between the client and the server, or at the server side, even in the case when the server is malicious.

Access Privacy. An intruder or a malicious server should not be able to perform statistical attacks by exploiting query patterns. For example, it should not be able to compromise data confidentiality by correlating known public information with frequently queried data items.

We will discuss how to design protocols that adapt to the existence of *trusted hardware* — so critical functionality can be delegated securely from clients to servers and increased assurance levels can be achieved more efficiently. Moreover, it is important to design for scalability to large data sets and high query throughputs. We note that client *authentication* and *authorization*, two important but orthogonal security dimensions, are extensively addressed in existing research, discussed in both this book and elsewhere [22, 27, 31, 33, 39, 68, 75, 79, 80, 88, 90, 102, 103, 105, 123]; therefore they and are not the main focus here. The assurances discussed here naturally complement these dimensions in providing increased end-to-end security.

2 Designing Secure Data Outsourcing Mechanisms.

2.1 Model

In our discourse, we will consider the following concise yet representative interaction model. Sensitive data is placed by a client on a database server managed by a *database service provider*. Later, the client or a third party will access the outsourced data through an online query interface exposed by the server. Network layer confidentiality is assured by mechanisms such as SSL/IPSec.

We will represent both the server and the client as interactive polynomial-time Turing Machines; we write Cli for the client and Serv for the server machine. A client can interact with the server and issue a sequence of update or processing queries $(Q_1, \ldots, Q_i)$. We call such a sequence of queries a *trace* $\mathcal{T}$. After executing a query Q, the client Turing Machine either outputs $\top$ or $\bot$, indicating whether the client accepts or rejects the server's response (denoted as $D_{\mathcal{T},Q}$); in the first case, the client believes that the server replied honestly. We write $\text{Cli}(\mathcal{T}, Q, D_{\mathcal{T},Q}) \in \{\top, \bot\}$ to denote the output of the client as a result of the server's execution of trace $\mathcal{T}$ and query Q yielding the result $D_{\mathcal{T},Q}$.

A server's response D is said to be *consistent* with both $\mathcal{T}$ and Q, if an honest server, after starting with an empty database and executing trace $\mathcal{T}$ honestly, would reply with D to the query Q. Two traces $\mathcal{T}$ and $\mathcal{T}'$ are called *similar* with

respect to Q, written as $\mathcal{T} \approx_Q \mathcal{T}'$, if the query Q yields the same answer when queried after a trace $\mathcal{T}$ or $\mathcal{T}'$, i.e., $D_{\mathcal{T},Q} = D_{\mathcal{T}',Q}$.

The data server is considered to be un-trusted, potentially malicious, compromised or simply faulty. Given the possibility to get away undetected, it will attempt to compromise data confidentiality, infer data access patterns and return incorrect query results. In certain cases we will assume reasonable computational limits such as the inability to factor large numbers or find cryptographic hash collisions. We will not make any limiting assumptions on the DBMS. In particular we will accommodate both multi-processor and distributed query processing DBMS. We will collaborate with other researches to investigate how to accommodate non-relational data integration [17] but mention that this does not constitute the subject of this work.

The main performance constraint we are interested in is *maintaining the benefits of outsourcing*. In particular, for a majority of considered operations, if they are more efficient (than client processing) in the unsecured data outsourcing model – then they should still be more efficient in its secured version. We believe this constraint is essential, as it is important to identify solutions that validate in real life.

We note the existence of a large number of apparently more elegant cryptographic primitives that could be deployed that would fail this constraint. In particular, experimental results indicate that often, individual data-item operations on the server should *not* involve any expensive modular arithmetic such as exponentiation or multiplication. We believe it is imperative to resist the (largely impractical) trend to use homomorphisms in server side operations unless absolutely necessary – as this often simplifies protocols in theory but fails in practice due to extremely poor performance, beyond usability.

Throughout this chapter we reference active secure hardware such as the IBM 4758 PCI [18] and the newer IBM 4764 PCI-X [19] cryptographic coprocessors [21]. The benefits of delpoying such hardware in un-trusted remote data processing contexts can be substantial, because the server can now run important parts of the secure client logic. Additionally, the secure hardware's proximity to the data will reduce communication overheads. Practical limitations of such devices however, make this a non-trivial task. To explain this, we briefly survey the processors.

The 4764 is a PowerPC - based board and runs embedded Linux. The 4758 is based on a Intel 486 architecture and is preloaded with a compact runtime environment that allows the loading of arbitrary external certified code. The CPUs can be custom programmed. Moreover, they (4758 models 2 and 23 and 4764 model 1) are compatible with the IBM Common Cryptographic Architecture (CCA) API [20]. The CCA implements common cryptographic services such as random number generation, key management, digital signatures, and encryption (DES/3DES,RSA). Both processors feature tamper resistant and responsive designs [56]. In the eventuality of illicit physical handling, the devices will simply destroy their internal state (in a process powered by internal long-term batteries) and then shutdown. Tamper resistant designs however, face major challenges in

heat dissipation. This is one of the main reasons why secure coprocessors are significantly constrained in both computation ability and memory (main heat producer) capacity, often being orders of magnitude slower that the main CPUs in their host systems. For example, at the higher end, the 4758s feature 100Mhz CPUs and 8MB+ of RAM.

These constraints require careful consideration in achieving efficient protocols. Simplistic implementations of query processors *inside* the SCPU are bound to fail in practice simply due to lack of performance. The host CPUs will remain starkly underutilized and the entire cost-proposition of having fast (unsecured) main CPUs and an expensive and slow secured CPU will be defeated. Efficient designs are likely to access the secure hardware just sparsely, in critical portions, not synchronized with the main data flow. Therefore we will pursue designs that use such hardware only as a trusted-aide, while considering its limited I/O and computation throughput. For example, we believe efficient solutions can be achieved by balancing a storage-computation trade-off when main un-secured storage capacity is significantly cheaper than the purchase of additional secure computation elements. In such a model, additional secure metadata structures are constructed over the outsourced data, by both clients and SCPUs. These enable the unsecured main CPU to perform computation-intensive portions of secure queries without requiring trusted hardware support. The cost of constructing these additional *helper* data structures will be amortized over multiple query instances.

We use the term *encryption* to denote any semantically secure (IND-CPA) encryption mechanism [65], unless specified otherwise. We note that the mechanisms introduced here do not depend on any specific encryption mechanism. A *one-way cryptographic hash* $\mathtt{H}()$ is a function with two important properties of interest: (i) it is computationally infeasible, for a given value $\mathtt{V}'$ to find a $\mathtt{V}$ such that $\mathtt{H}(\mathtt{V}) = \mathtt{V}'$ (one-wayness), and (ii) changing even one bit of the hash input causes random changes to the output bits (i.e., roughly half of them change even if one bit of the input is flipped). Examples of potential candidates are the MD5 (fast) or the SHA class of hashes (more secure). *Bloom filters* [35] offer a compact statistical representation of a set of data items and fast set inclusion tests. They are *one-way*, in that, the "contained" set items cannot be enumerated easily. For more details see [65, 113].

2.2 Query Correctness

Informally, we will call a query mechanism *correct* if the server is bound to the sequence of update requests performed by the client. Either the server responds correctly to a query or its malicious behavior is immediately detected by the client:

Definition 1. *A query protocol is correct, if (except with* negligible *probability [65]) for all traces $\mathcal{T}$ and $\mathcal{T}'$ with $\mathcal{T}' \not\approx_Q \mathcal{T}$, any query Q and server response $D_{\mathcal{T}',Q}$, we have* $\textsc{Cli}(\mathcal{T}, Q, D_{\mathcal{T}',Q}) = \perp$.

In applied settings, *correctness* in database outsourcing can be often decomposed into two protocol properties, namely *data integrity* and *query completeness*. Data integrity guarantees that outsourced data sets are not tampered with by the server. Completeness ensures that queries are executed against their entire target data sets and that query results are not 'truncated" by servers.

Existing work focuses mostly on solutions for simple one-dimensional range queries, and variants thereof. In a publisher-subscriber model, Devanbu et al. deployed Merkle trees to authenticate data published at a third party's site [54], and then explored a general model for authenticating data structures [97, 98]. Hard-to-forge verification objects are provided by publishers to prove the authenticity and provenance of query results.

In [104], mechanisms for efficient integrity and origin authentication for simple selection predicate query results are introduced. Different signature schemes (DSA, RSA, Merkle trees [100] and BGLS [37]) are explored as potential alternatives for data authentication primitives. Mykletun et al. [57] introduce *signature immutability* for aggregate signature schemes – the difficulty of computing new valid aggregated signatures from an existing set. Such a property is defeating a frequent querier that could eventually gather enough signatures data to answer other (un-posed) queries. The authors explore the applicability of signature-aggregation schemes for efficient data authentication and integrity of outsourced data. The considered query types are simple selection queries.

Similarly, in [94], digital signature and aggregation and chaining mechanisms are deployed to authenticate simple selection and projection operators. While these are important to consider, nevertheless, their expressiveness is limited. A more comprehensive, query-independent approach is desirable. Moreover, the use of strong cryptography renders this approach less useful. Often simply transferring the data to the client side will be faster.

In [108] *verification objects* VO are deployed to authenticate simple data retrieval in "edge computing" scenarios, where application logic and data is pushed to the edge of the network, with the aim of improving availability and scalability. Lack of trust in edge servers mandates validation for their results – achieved through verification objects.

In [77] Merkle tree and cryptographic hashing constructs are deployed to authenticate the result of simple range queries in a publishing scenario in which data owners delegate the role of satisfying user queries to a third-party untrusted publisher. Additionally, in [95] virtually identical mechanisms are deployed in database outsourcing scenarios. [53] proposes an approach for signing XML documents allowing untrusted servers to answer certain types of path and selection queries.

Drawbacks of these efforts include the fact that they operate in an unrealistic "semi - honest" adversarial model. As a result, for example, data updates are not handled properly and the mechanisms are vulnerable to "universe split" attacks discussed in section 2.2.

Moreover, deploying expensive cryptographic operations (e.g., aggregate signatures, homomorphisms) has the potential to defeat the very purpose of out-

sourcing. Unless the actual query predicates are comparably compute intensive, often simply transferring the *entire* database and executing the query on the client will be faster. This is the case simply because securely server - processing a bit will be more expensive that the bit transfer over a network. A detailed argument can be found in [118] and in section 2.4. Maybe most importantly, existing solutions operate under un-realistic "cooperating" server assumptions. For example, they are unable to address data updates. More specifically, at the time of a client update, the server is assumed to *cooperate* in also updating corresponding server-side security checksums and signature chains. A truly malicious server however, can choose to ignore such requests and compromise future correctness assurances by omitting the updated data from the results (causing an "universe split"). This drastically limits the applicability of these mechanisms.

We started to explore query correctness by first considering the query expressiveness problem. Thus, in [114] we proposed a novel method for proofs of *actual* query execution in an outsourced database framework for *arbitrary* queries. The solution prevents a "lazy" or malicious server from incompletely (or not at all) executing queries submitted by clients. It is based on a mechanism of runtime query "proofs" in a challenge - response protocol. For each batch of client queries, the server is "challenged" to provide a *proof of query execution* that offers assurance that the queries were actually executed with completeness, over their entire target data set. This proof is then checked at the client site as a prerequisite to accepting the actual query results as accurate.

The execution proofs are built around an extension to the *ringer* concept first introduced in [67]. Its core strength derives from the non-"invertibility" of cryptographic hash functions. In other words, a successful fake execution proof requires the "inversion"[1] of a cryptographic hash or a lucky guess. The probability of the lucky guess is known, controllable and can be made arbitrary small. If, as part of the response to a query execution batch, the server includes a correct, verifiable query execution proof, the client is provided with a (tunable) high level of assurance that the queries in the batch were executed correctly. This constitutes a strong counter-incentive to "lazy", (e.g., cost-cutting) behavior.

We implemented a proof of concept and experimentally validated it in a real-world data mining application, proving its deployment feasibility. We analyzed the solution and show that its overheads are reasonable and far outweighed by the added security benefits. For example an assurance level of over 95% can be achieved with less than 25% execution time overhead.

Future Work: Powerful Adversary. Arbitrary Queries. Data Updates.

As the above query execution proofs only validate server-side *processing* but not also actual returned results, handling truly malicious adversaries will require different mechanisms. Moreover, while compute-intensive query scenarios are extremely relevant in data-mining applications, a more general solution should consider general types of queries with less computation load per data tuple (e.g., aggregates such as SUM, COUNT). Handling these is especially challenging due

[1] We informally define "inversion" of hash functions as finding at least one input that hashes to a target output.

to the large size of the query space, the hardness of building general purpose authenticators and the hardness of predicting future query loads.

We believe future work should focus on two research directions: (1) the design of secure query (de)composition techniques coupled with specialized query - specific metadata that enables correctness assurance protocols for a set of primitive queries, and (2) mechanisms for trusted hardware.

In (1), additional server-side storage will be traded for efficient correctness assurances. At outsourcing time, in a pre-processing phase, clients generate query and predicate - specific metadata that will be stored on the server, authenticated by minimal state information maintained by clients. For each considered primitive predicate and type of query (e.g., simple range query), its corresponding "correctness metadata" will allow the client (or a trusted proxy such as a secure CPU) to assess the correctness of individual results. We call such primitive queries for which correctness can be assessed, "correctness-assured".

It is important to build on existing work [57, 77, 94, 95, 104], to reduce the computational footprint on the server, and allow consistent handling of updates in the presence of a truly malicious server. For example, we believe incremental hashing paradigms can be deployed to persist client-side authentication information. This will allow a client to efficiently authenticate returned signature values, thus detecting any malicious behavior even after updates.

Another future work item will be to design techniques that decompose or rewrite complex queries into a subset of the primitive queries considered above. Consider the following simple, yet illustrative query listing all account holders with account rates less than the Federal Reserve's base rate on January 1st, 2006:

```
SELECT accounts.name FROM accounts WHERE accounts.rate <
 (SELECT federalreserve.baserate FROM federalreserve
  WHERE convert(char(10),federalreserve.date,101)='01/01/2006')
```

Its correctness can be efficiently assessed by requiring the server to prove correctness for the inner query first, followed by the outer query. Similar decompositions can be applied to any correctness-assured nested queries. Nevertheless, often such query decomposition or rewriting cannot be achieved with efficiency for arbitrary queries in fully unsecured environments. For example, it is not trivial to extend correctness - assured simple range predicates to even marginally more complex multi-dimensional range queries such as

```
SELECT X.a FROM X WHERE X.b > 10 AND X.c > 20
```

It is important to investigate composition mechanisms that allow the utilization of metadata ensuring correctness of either simple range predicate (e.g., `X.b` $>$ `10` or `X.c` $>$ `20`), to guarantee correctness for the composite predicate.

To achieve correctness assurances for a larger class of queries we propose to consider mechanisms that leverage the presence of active secure hardware such as secure co-processors (SCPUs). Achieving efficiency however, is an extremely challenging task. Trivially deploying query processor functionality inside power

- constrained SCPUs is simply not scalable in practice due to limited communication and computation throughputs. We believe protocols that combine the query decomposition approach in (1) with SCPU processing for required, yet unavailable correctness-assured primitive queries constitute a promising avenue of future research. As a result, SCPU processing will be minimal and amortized over multiple query instances.

As an example, in the above multi-dimensional range query, a trusted SCPU hosted by the server will instruct the main server CPU to execute and prove correctness for the first predicate (`X.b` $>$ `10`) and then evaluate the second predicate (`X.c` $>$ `20`) securely on the result. Heuristics could be deployed to evaluate which of the individual predicates would result in a smaller result set so as to minimize the SCPU computation. Optionally, the process will also generate associated metadata for the joint predicate and cache it on the server for future use, effectively amortizing the cost of this query over multiple instances.

Operating in an *unified client model* [54,104] assumes the existence of a single client accessing the data store at any one time. In multi-threaded data-intensive application scenarios however, such a model is often of limited applicability. It is important to allow multiple client instances or even different parties to simultaneously access outsourced data sets.

This is challenging because allowing different parties to access the same data store may require the sharing of secrets among them. This is often not a scalable proposition, in particular considering different administrative domains. Moreover, data updates require special consideration in such a scenario due to what we call the *"universe split" phenomenon*. We explain this in the following.

In single - client settings, to efficiently handle incoming data updates, updateable metadata structures can be designed, e.g., leveraging such mechanisms as the incremental hashing paradigm of Bellare and Micciancio [26]. Recently we have demonstrated the feasibility of such methods in the framework of network data storage. In [117] outsourced documents were incrementally authenticated with efficient checksums allowing updates, document additions and removals in constant time.

However, when two clients simultaneously access the same data sets, a malicious server can chose to present to each client a customized version of the data universe, by keeping the other client's updates hidden from the current view. We believe other authors have encountered this issue in different settings, e.g., by Li et al. [91] in an un-trusted networked file system setting[2]. Naturally, if mutually aware of their accesses, the clients can use an external authenticated channel to exchange transactional state on each other's updates. This can occur either during their access, if simultaneous, or asynchronously otherwise. Periodically executing such exchanges will significantly decrease the probability of undetected illicit "universe split" server behavior. Over multiple transactions, undetected malicious behavior will become unsustainable.

In practice, such awareness and online interaction assumptions are not always acceptable, and often the only potential point of contact between clients is

[2] In their work universe splitting would be the inverse of "fetch-consistency"

the database server itself. One solution to this problem is to design alternative protocols that leverage the existence of active secure hardware such as secure co-processors (SCPU). The SCPU will authenticate clients securely and also persist transactional state, including a minimal amount of checksum information used to authenticate transaction chains of committed client updates. The unique vendor-provided SCPU public key and its associated trust chain provide an authenticated communication channel between the SCPU and database clients. The clients will use this channel to retrieve up to date transactional state at the initiation of each server interaction. This will defeat "universe split" attacks. Servers are unable to impersonate SCPUs without access to the secrets in its tamper-proof storage.

2.3 Data Confidentiality

Confidentiality constitutes another essential security dimension required in data outsourcing scenarios, especially when considering sensitive information. Potentially un-trusted servers should be able to process queries on encrypted data on behalf of clients without compromising confidentiality. To become practical, any such processing mechanism requires a certain level of query expressiveness. For example, allowing only simple data retrieval queries will often not be sufficient to justify the outsourcing of the data – the database would then be used as a passive data repository. We believe it is important to efficiently support complex queries such as joins and aggregates with confidentiality and correctness.

Hacigumus et al. [71] propose a method to execute SQL queries over partly obfuscated outsourced data. The data is divided into secret partitions and queries over the original data can be rewritten in terms of the resulting partition identifiers; the server can then partly perform queries directly. The information leaked to the server is claimed to be 1-out-of-s where s is the partition size. This balances a trade-off between client-side and server-side processing, as a function of the data segment size. At one extreme, privacy is completely compromised (small segment sizes) but client processing is minimal. At the other extreme, a high level of privacy can be attained at the expense of the client processing the queries in their entirety. Moreover, in [76] the authors explore optimal bucket sizes for certain range queries. Similarly, data partitioning is deployed in building "almost"-private indexes on attributes considered sensitive. An untrusted server is then able to execute "obfuscated range queries with minimal information leakage". An associated privacy-utility trade-off for the index is discussed. As detailed further in section 2.3 the main drawbacks of these solutions lies in their computational impracticality and inability to provide strong confidentiality.

One of the main drawbacks of such mechanisms is the fact that they leak information to the server, at a level corresponding to the granularity of the partitioning function. For example, if such partitioning is used in a range query, to execute rewritten queries at the partition level, the server will be required to precisely know the range of values that each partition contains. Naturally, increasing partition sizes tends to render this knowledge more fuzzy. This, however, requires additional client side work in pruning the (now) larger results (due

to the larger partitions). Even if a single data tuple matches the query, its entire corresponding partition will be transferred to the client. On the other hand, reducing partition size will immediately reveal more information to the server, as the smaller number of items per partition and the knowledge of the covered range will allow it to determine more accurately what the likely values are for each tuple. Additionally, for more complex queries, particularly joins, due to the large segments, such methods can feature an communication overhead larger than the entire database, hardly a practical proposition.

Nevertheless, these efforts illustrate a trade-off between confidentiality and overheads: large partitions reveal less but require more computation on the client, small partitions reveal more but increase efficiency. Ultimately, however, unless partitions are very large (in which case the purpose of outsourcing is likely defeated by the additional overheads) true confidentiality cannot be achieved by such partitioning schemes. Statistical security needs to be replaced by efficient, yet stronger mechanisms. In the following we show how this can be achieved not only for range queries but also for more complex joins.

In ongoing work [42] we explore a low-overhead method for executing binary predicate joins with confidentiality on outsourced data. It handles *general* binary join predicates that satisfy certain properties: for any value in the considered data domain, the *number* of corresponding "matching" pair values (for which the predicate holds) is (i) finite, and (ii) the average of its expected value is upper bound. We call these predicates *expected finite match* (EFM) predicates.

Such predicates are extremely common and useful, including discrete data scenarios, such as ranges, inventory and company asset data-sets, forensics, genome and DNA data (e.g., fuzzy and exact Hamming distances), and healthcare databases (e.g., bacteria to antibiotics matches). For illustration purposes let us consider the following discrete time – range join query that joins arrivals with departures within the same hour (e.g., in a train station):

```
SELECT *  FROM arrivals,departures
WHERE departures.time - arrivals.time < 60
```

For any finite time granularity (e.g. minutes) the join predicate above is an EFM predicate (e.g., with an AEMS of 60). Performing such joins at the server side on encrypted data, is the main functionality desired here.

To analyze the *confidentiality assurances* of this solution we will consider here a server that is *curious*: given the possibility to get away undetected, it will attempt to compromise data confidentiality (e.g., in the process of query execution). Naturally, it should **not** be able to evaluate predicates (i) without the permission of the client, (ii) on two values of the same attribute, and (iii) on data not specified/allowed by the client – specifically, no inter-attribute transitivity should be possible. Additionally it should not be able to (iv) evaluate other predicates on "unlocked" data. This also means that no additional information should be leaked in the process of predicate evaluation. For example, allowing the evaluation of $p(x, y) := (|x - y| < 100)$, should not reveal $|x - y|$.

One solution relies on the use of predicate-specific metadata that clients place on the server together with the main data sets. This metadata does not

reveal anything about the main data fields and stays in a "locked" state until its corresponding data is involved in a join. The client then provides "unlocking" information for the metadata and the server is able to perform *exactly* the considered query, without finding out any additional information. In the following we briefly outline this. For more details see [42].

Let N be a public security parameter, and K a symmetric (semantically secure) encryption key. For each column A, let $R_1^A \neq R_2^A$ be two random uniform values in $\{0,1\}^N$. In a client *pre-processing* phase, for each confidential data attribute A with elements a_i, $i = 1..n$, the client computes an obfuscation of a_i, $O(a_i) := H(a_i) \oplus R_1^A$. For all values $y \in P(a_i) := \{y | p(a_i, y) = true\}$, the client computes $H(y) \oplus R_2^A$. and stores it into a Bloom filter specific to a_i, $BF(a_i)$. It then outsources $\{E_K(a_i),\ O(a_i),\ BF(a_i)\}$ to the server. To allow a join of two columns A and B on the predicate p, the client sends the server the value $q_{AB} = R_2^A \oplus R_1^B$. For each element a_i in column A and b_j in column B, the server computes $T_{b \to a} := O(b_j) \oplus q_{AB} = H(b_j) \oplus R_2^A$. It then outputs all tuples $< E_K(a_i), E_K(b_j), \ldots >$ for which $BF(a_i)$ contains $T_{b \to a}$. The following can be shown:

Theorem 1. *The server cannot perform join operations on initially stored data.*

Theorem 2. *The server cannot perform transitive joins.*

Theorem 3. *Given a binary EFM predicate p, for any matching pair of values returned as a result of a join, $< x' = E_K(a_i), y' = E_K(b_j) >$, no additional information about a_i and b_j or their relationship can be inferred by the server, other than the fact that $p(a_i, b_j) = true$.*

The solution handles *data updates* naturally. For any new incoming data item, the client pre-processing can be executed per-item and its results simply forwarded to the server. Additionally, in the case of a multi-threaded server, multiple clients (sharing secrets and keys) can access the same data store simultaneously.

We note also that *multiple predicate evaluations* are also accommodated naturally. Confidentiality can be provided for the attributes involved in binary EFM predicates. In the following database schema, the association between patients and diseases is confidential but any other information is public and can be used in joins. To return a list of New York City patient names and their associated antibiotics (but not their disease) the server will access both confidential (disease) and non-confidential (name,zip-code) values. In the following, only the predicate $md()$ – associating antibiotics with diseases – will operate on confidential data:

```
SELECT patients.name,antibiotics.name FROM patients,antibiotics
WHERE md(patients.disease,antibiotics.name) AND patients.zipcode = 10128
```

This will be achieved (as discussed above) by encrypting the `patients.disease` attribute and generating metadata for the `antibiotics` relation (which contains a list of diseases that each antibiotic is recommended for).

Additional predicate instances and applications of this solution are explored in [42], including mechanisms for Hamming distance evaluations and DNA fuzzy

match predicates. Moreover, we show that the computation overheads of the solution are small. In initial evaluations, throughputs of well beyond 0.5 million predicate evaluations per second can be accomodated.

Future Work: Arbitrary Predicates. Policies. Query Composability.

In future work, we believe it is important to pursue arbitrary query types and multi-assurance compositions. For example we would like to understand how to endow the above method with correctness assurances and data access privacy as discussed in sections 2.2, 2.4 respectively.

Moreover, it is important to analyze the applicability of the protocols for general types of predicates. We believe a recursive decomposition approach can be applied to handle multiple argument EFM predicates. Transformations from arbitrary predicates to a canonical EFM form should be explored. In a first stage this is easy to achieve by simply discretizing queries over continuous data domains. As this will introduce small errors in results (of a magnitude inverse proportional to the quantization), this process needs to be designed such that the errors will result only in the *addition* of a small, controllable, number of non-matching tuples. These will then be pruned by the client.

To fully leverage the potential offered by confidentiality assurances, it is important to investigate an integration with security policy frameworks [60,111]. This will allow for more complex specifications over the space of data sets, access rights, confidentiality policies and principals. For example, such specifications could include relaxation of expensive DBMS - maintained access control for data sets that are already encrypted.

Exploring novel notions of confidential query "composability" in the presence of multiple confidential data sources and associated secrets (e.g., cryptographic keys) is another avenue of future research. We believe this can be achieved by deploying intra-server secure multi-party computation (SMC) protocols [55, 58, 59, 63, 78] mediated by secure hardware. The presence of secure hardware will result in more efficient, practical SMC. This will ultimately allow for multi-source confidential data integration.

2.4 Data Access Privacy

In existing protocols, even though data sets are stored in encrypted form on the server, the *client query access patterns* leak essential information about the data. A simple attack can correlate known public information with *hot* data items (i.e., with high access rates), effectively compromising their confidential nature. In competing business scenarios, such leaks can be extremely damaging, particularly due to their unpredictable nature.

This is why, to protect confidentiality, it is important to also provide assurances of access pattern privacy. No existing work has tackled this problem yet for relational frameworks. It is thus essential to explore query protocols that leak minimal information about the currently executing query. Access patterns to data tuples become less meaningful when access semantics are unknown to the server. For example the binary predicate join method proposed above does

not require the server to know the actual join predicates. Achieving such goals for *arbitrary* relational queries will be a challenging proposition in today's query processors, potentially requiring fundamental changes in base query processing.

To achieve these goals we first turn to existing research. *Private Information Retrieval* (PIR) protocols were first proposed as a theoretical primitive for accessing individual items of outsourced data, while preventing servers to learn anything about the client's access patterns [47]. Chor et al. [48] proved that in information theoretic settings in which queries do not reveal any information about the accessed data items, a solution requires $\Omega(n)$ bits of communication. To avoid this overhead, they show that for multiple *non-colluding* databases holding replicated copies of the data, PIR schemes exist that require only sub-linear communication overheads. This multi-server assumption however, is rarely viable in practice.

In single-server settings, it is known that PIR requires a full transfer of the database [47, 49] for computationally unbounded servers. For bounded adversaries however, *computational* PIR (cPIR) mechanisms have been proposed [40, 41, 45, 86, 87, 93, 96, 122]. In such settings however, it is trivial to establish an $O(n)$ lower bound on server processing, mandating expensive trapdoor operations per bit, to achieve access privacy. This creates a significant privacy - efficiency trade-off between the required server computation cycles and the time to actually transfer the data and perform the query at the client site.

We explore this trade-off in [118] where we discuss single-server computational PIR *for the purpose of preserving client access patterns leakage*. We show that deployment of non-trivial single server private information retrieval protocols on real (Turing) hardware is orders of magnitude more time-consuming than trivially transferring the entire database to the client. The deployment of computational PIR in fact *increases* both overall execution time, and the probability of *forward* leakage, when the deployed present trapdoors become eventually vulnerable – e.g., today's access patterns will be revealed once factoring of today's values will become possible in the future.

We note that these results are beyond existing knowledge of mere "impracticality" under unfavorable assumptions. On real hardware, *no* existing non-trivial single server PIR protocol could have possibly had outperformed the trivial client-to-server transfer of records in the past, and is likely not to do so in the future either. Informally, this is due to the fact that it is more expensive to PIR-process one bit of information than to transfer it over a network.

PIR's aim is to simply transfer one single remote bit with privacy. We showed above that theoretical lower bounds prevent current cryptography to offer efficient solutions in practical settings. Arguably, for more complex query processing this will also be the case. Thus it is important to design practical solutions that have the potential to break the PIR computation-privacy trade-off. We believe a very promising avenue for further research relies on deploying secure hardware hosted by the server, allowing the delegation of client-logic in closer data proximity.

And because (as discussed above) trivial "run client "proxy" inside secure CPU" approaches are likely to be impractical – as typically such hardware is orders of magnitude slower than main CPUs – any solution needs to deploy SCPUs efficiently, to defeat statistical correlation attacks on data access patterns.

3 Related Work.

Extensive research has focused on various aspects of DBMS security, including access control techniques as well as general information security issues [29, 31, 51, 73, 75, 80, 81, 90, 106, 107, 110, 112], many of which are discussed elsewhere in this book. Additionally, increasing awareness of requirements for data storage security mechanisms and support can be found with DBMS vendors such as IBM [10] and Oracle [16].

3.1 Database as a Service

The paradigm of providing a database as a service recently emerged [72] as a viable alternative, likely due in no small part to the dramatically increasing availability of fast, cheap networks. Given the global, networked, possibly hostile nature of the operation environments, security assurances are paramount.

Data Sharing. Statistical and *Hippocratic* databases aim to address the problem of allowing aggregate queries on confidential data (stored on *trusted* servers) without additional information leaks [24, 25, 50, 51, 89] to the queries. In [125] Zhang et al. discuss privacy in information sharing scenarios in a distributed multi-party context, where each party operates a private database. An leakage measure is defined for information sharing and several privacy multi-party protocols deploying commutative encryption are defined.

3.2 XML Sharing

In [30] Bertino et al. discuss a solution for access control to XML data. They deploy multi-key encryption such that only the appropriate parts of outsourced XML documents can be accessed by principals. In [32] (also in [28]), they propose a mechanism deploying watermarking [23, 69, 92, 115, 116, 120] to protect ownership for outsourced medical data. Similarly, Carminati et al. ensure the confidentiality of XML in a distributed peer network by using access rights and encryption keys associated with XML nodes [43]. They enforce the authenticity and integrity of query answers using Merkle signatures [100]. This complicates outsourcing of new documents as new Merkle trees will need to be generated. To ensure query correctness, the server also stores encrypted query templates containing the structure of the original documents. This solution is insecure because it leaks decryption keys and content access patterns.

3.3 Secure Storage

Encrypted Storage. Blaze's CFS [34], TCFS [44], EFS [101], StegFS [99], and NCryptfs [124] are file systems that encrypt data before writing to stable storage. NCryptfs is implemented as a layered file system [74] and is capable of being used even over network file systems such as NFS. SFS [70] and BestCrypt [82] are device driver level encryption systems. Encryption file systems and device drivers protect the confidentiality of data, but do not allow for efficient queries, search, correctness, or access privacy assurances.

Integrity-Assured Storage. Tripwire [84, 85] is a user level tool that verifies stored file integrity at scheduled intervals of time. File systems such as I^3FS [83], GFS [62], and Checksummed NCryptfs [119] perform online real-time integrity verification. Venti [109] is an archival storage system that performs integrity assurance on read-only data. SUNDR [91] is a network file system designed to store data securely on untrusted servers and allow clients to detect unauthorized accesses as long as they see each other's file modifications.

3.4 Searches on Encrypted Data

Song et al. [121] propose a scheme for performing simple keyword search on encrypted data in a scenario where a mobile, bandwidth-restricted user wishes to store data on an untrusted server. The scheme requires the user to split the data into fixed-size words and perform encryption and other transformations. Drawbacks of this scheme include fixing the size of words, the complexities of encryption and search, the inability of this approach to support access pattern privacy, or retrieval correctness. Eu-Jin Goh [64] proposes to associate indexes with documents stored on a server. A document's index is a Bloom filter [35] containing a codeword for each unique word in the document. Chang and Mitzenmacher [46] propose a similar approach, where the index associated with documents consists of a string of bits of length equal to the total number of words used (dictionary size). Boneh et al. [36] proposed an alternative for senders to encrypt e-mails with recipients' public keys, and store this email on untrusted mail servers. They present two search protocols: (1) a non-interactive search-able encryption scheme based on a variant of the Diffie-Hellman problem that uses bilinear maps on elliptic curves; and (2) a protocol using only trapdoor permutations, requiring a large number of public-private key pairs. Both protocols are computationally expensive. Golle et al. [66] extend the above idea to conjunctive keyword searches on encrypted data. They propose two solutions. (1) The server stores capabilities for conjunctive queries, with sizes linear in the total number of documents. They claim that a majority of the capabilities can be transferred offline to the server, under the assumption that the client knows beforehand its future conjunctive queries. (2) Doubling the size of the data stored by the server, which reduces the communication overheads between clients and servers significantly. The scheme requires users to specify the exact positions where the search matches have to occur, and hence is impractical. Brinkman et al. [38] deploy secret splitting of polynomial expressions to search in encrypted XML.

References

1. Activehost.com Internet Services. Online at http://www.activehost.com.
2. Adhost.com MySQL Hosting. Online at http://www.adhost.com.
3. Alentus.com Database Hosting. Online at http://www.alentus.com.
4. Datapipe.com Managed Hosting Services. Online at http://www.datapipe.com.
5. Discountasp.net Microsoft SQL Hosting. Online at http://www.discountasp.net.
6. Gate.com Database Hosting Services. Online at http://www.gate.com.
7. Hostchart.com Web Hosting Resource Center. Online at http://www.hostchart.com.
8. Hostdepartment.com MySQL Database Hosting. Online at http://www.hostdepartment.com/mysqlwebhosting/.
9. IBM Data Center Outsourcing Services. Online at http://www-1.ibm.com/services/.
10. IBM Data Encryption for DB2. Online at http://www.ibm.com/software/data/db2.
11. Inetu.net Managed Database Hosting. Online at http://www.inetu.net.
12. Mercurytechnology.com Managed Services for Oracle Systems. Online at http://www.mercurytechnology.com.
13. Neospire.net Managed Hosting for Corporate E-business. Online at http://www.neospire.net.
14. Netnation.com Microsoft SQL Hosting. Online at http://www.netnation.com.
15. Opendb.com Web Database Hosting. Online at http://www.opendb.com.
16. Oracle: Database Encryption in Oracle 10g. Online at http://www.oracle.com/database.
17. The IBM WebSphere Information Integrator. Online at http://www.ibm.com/software/data/integration.
18. IBM 4758 PCI Cryptographic Coprocessor. Online at http://www-03.ibm.com/security/cryptocards/pcicc/overview.shtml, 2006.
19. IBM 4764 PCI-X Cryptographic Coprocessor (PCIXCC). Online at http://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml, 2006.
20. IBM Common Cryptographic Architecture (CCA) API. Online at http://www-03.ibm.com/security/cryptocards//pcixcc/overcca.shtml, 2006.
21. IBM Cryptographic Hardware. Online at http://www-03.ibm.com/security/products/, 2006.
22. Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst.*, 15(4):706–734, 1993.
23. Andre Adelsbach and Ahmad Sadeghi. Advanced techniques for dispute resolving and authorship proofs on digital works. In *Proceedings of SPIE Electronic Imaging*, 2003.
24. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *Proceedings of the International Conference on Very Large Databases VLDB*, pages 143–154, 2002.
25. Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD*, pages 439–450, 2000.
26. M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Proceedings of EuroCrypt*, 1997.

27. Steven M. Bellovin. Spamming, phishing, authentication, and privacy. *Communications of the ACM*, 47(12):144, 2004.
28. E. Bertino. Data hiding and security in an object-oriented database system. In *Proceedings of the 8th IEEE International Conference on Data Engineering*, 1992.
29. Elisa Bertino, M. Braun, Silvana Castano, Elena Ferrari, and Marco Mesiti. Author-X: A Java-Based System for XML Data Protection. In *IFIP Workshop on Database Security*, pages 15–26, 2000.
30. Elisa Bertino, Barbara Carminati, and Elena Ferrari. A temporal key management scheme for secure broadcasting of xml documents. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 31–40, 2002.
31. Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems*, 17(2), 1999.
32. Elisa Bertino, Beng Chin Ooi, Yanjiang Yang, and Robert H. Deng. Privacy and ownership preserving of outsourced medical data. In *Proceedings of the International Conference on Data Engineering*, 2005.
33. Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. The kryptoknight family of light-weight protocols for authentication and key distribution. *IEEE/ACM Trans. Netw.*, 3(1):31–41, 1995.
34. M. Blaze. A Cryptographic File System for Unix. In *Proceedings of the first ACM Conference on Computer and Communications Security*, pages 9–16, Fairfax, VA, 1993. ACM.
35. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
36. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proceedings of Eurocrypt 2004*, pages 506–522. LNCS 3027, 2004.
37. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EuroCrypt*, 2003.
38. R. Brinkman, J. Doumen, and W. Jonker. Using secret sharing for searching in encrypted data. In *Secure Data Management*, 2004.
39. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles*, pages 1–13, New York, NY, USA, 1989. ACM Press.
40. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylog communication. In *Proceedings of EUROCRYPT*, 1999.
41. C. Cachin, S. Micali, and M. Stadler. Private Information Retrieval with Polylogarithmic Communication. In *Proceedings of Eurocrypt*, pages 402–414. Springer-Verlag, 1999.
42. Bogdan Carbunar and Radu Sion. Arbitrary-Predicate Joins for Outsourced Data with Privacy Assurances, 2006. Stony Brook Network Security and Applied Cryptography Lab Tech Report 2006-07.
43. B. Carminati, E. Ferrari, and E. Bertino. Assuring security properties in third-party architectures. In *Proceedings of International Conference on Data Engineering (ICDE)*, 2005.
44. G. Cattaneo, L. Catuogno, A. Del Sorbo, and P. Persiano. The Design and Implementation of a Transparent Cryptographic Filesystem for UNIX. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 245–252, Boston, MA, June 2001.

45. Y. Chang. Single-Database Private Information Retrieval with Logarithmic Communication. In *Proceedings of the 9th Australasian Conference on Information Security and Privacy ACISP*. Springer-Verlag, 2004.
46. Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. Cryptology ePrint Archive, Report 2004/051, 2004. `http://eprint.iacr.org/`.
47. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.
48. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of FOCS*. IEEE Computer Society, 1995.
49. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
50. Chris Clifton, Murat Kantarcioglu, AnHai Doan, Gunther Schadow, Jaideep Vaidya, Ahmed Elmagarmid, and Dan Suciu. Privacy-preserving data integration and sharing. In *The 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 19–26. ACM Press, 2004.
51. Chris Clifton and Don Marks. Security and privacy implications of data mining. In *Workshop on Data Mining and Knowledge Discovery*, pages 15–19, Montreal, Canada, 1996. Computer Sciences, University of British Columbia.
52. CNN. Feds seek Google records in porn probe. Online at `http://www.cnn.com`, January 2006.
53. Premkumar T. Devanbu, Michael Gertz, April Kwong, Chip Martel, G. Nuckolls, and Stuart G. Stubblebine. Flexible authentication of XML documents. In *ACM Conference on Computer and Communications Security*, pages 136–145, 2001.
54. Premkumar T. Devanbu, Michael Gertz, Chip Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In *IFIP Workshop on Database Security*, pages 101–112, 2000.
55. W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In *Proceedings of the 1st ACM Workshop on Security and Privacy in E-Commerce*, 2000.
56. Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert van Doorn, Sean W. Smith, and Steve Weingart. Building the ibm 4758 secure coprocessor. *Computer*, 34(10):57–66, 2001.
57. Einar Mykletun and Maithili Narasimha and Gene Tsudik. Signature Bouquets: Immutability for Aggregated/Condensed Signatures. In *Proceedings of the European Symposium on Research in Computer Security ESORICS*, pages 160–176, 2004.
58. Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. In *ICALP '01: Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, pages 927–938, 2001.
59. M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *In Advances in Cryptology EUROCRYPT*, pages 1–19, 2004.
60. Irini Fundulaki and Maarten Marx. Specifying access control policies for xml documents with xpath. In *The ACM Symposium on Access Control Models and Technologies*, pages 61–69. ACM Press, 2004.
61. Gartner, Inc. Server Storage and RAID Worldwide. Technical report, Gartner Group/Dataquest, 1999. `www.gartner.com`.

62. S. Ghemawat, H. Gobioff, and S. T. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29–43, Bolton Landing, NY, October 2003. ACM SIGOPS.
63. Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikinen. On private scalar product computation for privacy-preserving data mining. In *ICISC*, pages 104–120, 2004.
64. E. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216/`.
65. O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.
66. P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In *Proceedings of ACNS*, pages 31–45. Springer-Verlag; Lecture Notes in Computer Science 3089, 2004.
67. Philippe Golle and Ilya Mironov. Uncheatable distributed computations. In *Proceedings of the 2001 Conference on Topics in Cryptology*, pages 425–440. Springer-Verlag, 2001.
68. Li Gong. Efficient network authentication protocols: lower bounds and optimal implementations. *Distrib. Comput.*, 9(3):131–145, 1995.
69. David Gross-Amblard. Query-preserving watermarking of relational databases and xml documents. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 191–201, New York, NY, USA, 2003. ACM Press.
70. P. C. Gutmann. Secure filesystem (SFS) for DOS/Windows. `www.cs.auckland.ac.nz/~pgut001/sfs/index.html`, 1994.
71. H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 216–227. ACM Press, 2002.
72. H. Hacigumus, B. R. Iyer, and S. Mehrotra. Providing database as a service. In *IEEE International Conference on Data Engineering (ICDE)*, 2002.
73. J. Hale, J. Threet, and S. Shenoi. A framework for high assurance security of distributed objects, 1997.
74. J. S. Heidemann and G. J. Popek. File system development with stackable layers. *ACM Transactions on Computer Systems*, 12(1):58–89, February 1994.
75. E. Hildebrandt and G. Saake. User Authentication in Multidatabase Systems. In R. R. Wagner, editor, *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications, August 26–28, 1998, Vienna, Austria*, pages 281–286, Los Alamitos, CA, 1998. IEEE Computer Society Press.
76. B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proceedings of ACM SIGMOD*, 2004.
77. HweeHwa Pang and Arpit Jain and Krithi Ramamritham and Kian-Lee Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *Proceedings of ACM SIGMOD*, 2005.
78. Piotr Indyk and David Woodruff. Private polylogarithmic approximations and efficient matching. In *Theory of Cryptography Conference*, 2006.
79. S. Jajodia, P. Samarati, and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy*, pages 31–42, Oakland, CA, May 04-07 1997. IEEE Press.
80. S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on Security and Privacy. Oakland, CA*, pages 31–42, 1997.

81. S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD*, 1997.
82. Jetico, Inc. BestCrypt software home page. `www.jetico.com`, 2002.
83. A. Kashyap, S. Patil, G. Sivathanu, and E. Zadok. I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System. In *Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004)*, pages 69–79, Atlanta, GA, November 2004. USENIX Association.
84. G. Kim and E. Spafford. Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection. In *Proceedings of the Usenix System Administration, Networking and Security (SANS III)*, 1994.
85. G. Kim and E. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *Proceedings of the 2nd ACM Conference on Computer Commuications and Society (CCS)*, November 1994.
86. E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings of FOCS*. IEEE Computer Society, 1997.
87. E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In *Proceedings of EUROCRYPT*, 2000.
88. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: theory and practice. *ACM Trans. Comput. Syst.*, 10(4):265–310, 1992.
89. Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David J. DeWitt. Limiting disclosure in hippocratic databases. In *Proceedings of the International Conference on Very Large Databases VLDB*, pages 108–119, 2004.
90. Li, Feigenbaum, and Grosof. A logic-based knowledge representation for authorization with delegation. In *PCSFW: Proceedings of the 12th Computer Security Foundations Workshop*, 1999.
91. J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure Untrusted Data Repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI 2004)*, pages 121–136, San Francisco, CA, December 2004. ACM SIGOPS.
92. Yingjiu Li, Vipin Swarup, and Sushil Jajodia. A robust watermarking scheme for relational data. In *Proceedings of the Workshop on Information Technology and Systems (WITS)*, pages 195–200, 2003.
93. H. Lipmaa. An oblivious transfer protocol with log-squared communication. Cryptology ePrint Archive, 2004.
94. Maithili Narasimha and Gene Tsudik. DSAC: integrity for outsourced databases with signature aggregation and chaining. Technical report, 2005.
95. Maithili Narasimha and Gene Tsudik. Authentication of Outsourced Databases using Signature Aggregation and Chaining. In *Proceedings of DASFAA*, 2006.
96. E. Mann. Private access to distributed information. Master's thesis, Technion - Israel Institute of Technology, 1998.
97. C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. A general model for authenticated data structures. Technical report, 2001.
98. Charles Martel, Glen Nuckolls, Premkumar Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.
99. A. D. McDonald and M. G. Kuhn. StegFS: A Steganographic File System for Linux. In *Information Hiding*, pages 462–477, 1999.

100. R. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Research in Security and Privacy*, 1980.
101. Microsoft Research. Encrypting File System for Windows 2000. Technical report, Microsoft Corporation, July 1999. `www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp`.
102. Fabian Monrose and Aviel D. Rubin. Authentication via keystroke dynamics. In *ACM Conference on Computer and Communications Security*, pages 48–56, 1997.
103. Fabian Monrose and Aviel D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351–359, 2000.
104. E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *ISOC Symposium on Network and Distributed Systems Security NDSS*, 2004.
105. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
106. M. Nyanchama and S. L. Osborn. Access rights administration in role-based security systems. In *Proceedings of the IFIP Workshop on Database Security*, pages 37–56, 1994.
107. Sylvia L. Osborn. Database security integration using role-based access control. In *Proceedings of the IFIP Workshop on Database Security*, pages 245–258, 2000.
108. HweeHwa Pang and Kian-Lee Tan. Authenticating query results in edge computing. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 560, Washington, DC, USA, 2004. IEEE Computer Society.
109. S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST 2002)*, pages 89–101, Monterey, CA, January 2002. USENIX Association.
110. David Rasikan, Sang H. Son, and Ravi Mukkamala. Supporting security requirements in multilevel real-time databases, citeseer.nj.nec.com/david95supporting.html, 1995.
111. Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 551–562. ACM Press, 2004.
112. Ravi S. Sandhu. On five definitions of data integrity. In *Proceedings of the IFIP Workshop on Database Security*, pages 257–267, 1993.
113. B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley & Sons, 1996.
114. Radu Sion. Query execution assurance for outsourced databases. In *Proceedings of the Very Large Databases Conference VLDB*, 2005.
115. Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Relational data rights protection through watermarking. *IEEE Transactions on Knowledge and Data Engineering TKDE*, 16(6), June 2004.
116. Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Ownership proofs for categorical data. *IEEE Transactions on Knowledge and Data Engineering TKDE*, 2005.
117. Radu Sion and Bogdan Carbunar. Indexed Keyword Search with Privacy and Query Completeness, 2005. Stony Brook Network Security and Applied Cryptography Lab Tech Report 2005-07.
118. Radu Sion and Bogdan Carbunar. On the Computational Practicality of Private Information Retrieval. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2007. Stony Brook Network Security and Applied Cryptography Lab Tech Report 2006-06.

119. G. Sivathanu, C. P. Wright, and E. Zadok. Enhancing File System Integrity Through Checksums. Technical Report FSL-04-04, Computer Science Department, Stony Brook University, May 2004. `www.fsl.cs.sunysb.edu/docs/nc-checksum-tr/nc-checksum.pdf`.
120. J. Smith and C. Dodge. Developments in steganography. In A. Pfitzmann, editor, *Proceedings of the third Int. Workshop on Information Hiding*, pages 77–87, Dresden, Germany, September 1999. Springer Verlag.
121. D. Xiaodong Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P 2000)*. IEEE Computer Society, 2000.
122. J. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *Proceedings of Asia Crypt*, pages 357–371, 1998.
123. Thomas Y. C. Woo and Simon S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, 1992.
124. C. P. Wright, M. Martino, and E. Zadok. NCryptfs: A Secure and Convenient Cryptographic File System. In *Proceedings of the Annual USENIX Technical Conference*, pages 197–210, San Antonio, TX, June 2003. USENIX Association.
125. Nan Zhang and Wei Zhao. Distributed Privacy Preserving Information Sharing. In *Proceedings of the International Conference on Very Large Databases VLDB*, 2005.