Secure Computation on Outsourced Data: A 10-year Retrospective

Hakan Hacıgümüş[1], Bala Iyer[2], and Sharad Mehrotra[3]

[1] NEC Labs America
hakanh@acm.org
[2] IBM Silicon Valley Lab.
balaiyer@us.ibm.com
[3] University of California, Irvine, USA
sharad@ics.uci.edu

Abstract. This paper outlines the "behind the scene" story of how we wrote the DASFAA 2004 paper on secure computation on outsourced data to support SQL aggregation queries. We then describe some of the research we have done following the paper.

1 Paper Background

This paper was a result of a successful collaboration between IBM Silicon Valley Laboratory and the University of California, Irvine that explored the idea of outsourcing database management to third-party service providers. At the time, service-based software architectures and software-as-a-service were emerging concepts. Such architectures, made possible by the advances in high-speed networking and ubiquitous connectivity, provided numerous advantages: it alleviated the need for organizations to purchase and deploy expensive software and hardware to run the software on, deal with software upgrades, hire professionals to help maintain the infrastructure etc. Instead, in the software as a service model (also known also, at the time, as the application service provider (ASP) model), the end-users could rent the software over the Internet and pay for what they used. The model was very attractive in the light of the fact that around that time, people costs were beginning to dominate the overall costs of IT solutions. The ASP model, helped bring the people costs down significantly since the task of administration and software maintenance was now outsourced to service providers bringing in economy of scales.

Our collaboration started on a fateful SIGMOD Conference meeting that brought authors of this paper together. The key question we asked was: Given the emerging software-as-a-service, what will it take to provide data management as a service (or the DAS model as we called it)? In the DAS model, the service provider would provide seamless mechanisms for organizations to create, store, access their databases. Moreover, the responsibility of data management, i.e. database backup, administration, restoration, database reorganization to reclaim space, migration to new versions without impacting availability, etc. will

S.S. Bhowmick et al. (Eds.): DASFAA 2014, Part I, LNCS 8421, pp. 16–27, 2014.

be taken over by service providers. Users wishing to access data will now access it using the hardware and software at the service provider instead of their own organizations computing infrastructure. The application will not be impacted due to outages in software, hardware, and networking changes or failures at the database service provider sites. The DAS model will alleviate the problem of purchasing, installing, maintaining and updating the software and administering the system. Instead, users will be able to use the ready system maintained by service provider for its database needs.

1.1 Technical Challenges

Intuitively, we felt that database-as-a-service (or DAS model as we called it) would inherit all the advantages of the software-as-a-service model – even more so, given that for organizations data management, given its complexity, is a significant portion of the overall IT expense. Furthermore, data management, at the time (as is still the case) has a significant people cost requiring professionals to create, implement, and tune data management solutions. Given, what we felt was a unique opportunity business opportunity; we explored new challenges that would arise if we were to adopt a service-oriented model for data management. Our first paper on the topic appeared in the ICDE 2002 conference that raised multiple challenges in supporting the DAS model. In particular, we identified three challenges: a) appropriate user interfaces through which database services and data is presented to the end-user - the interface must be easy to use, yet be sufficiently powerful to allow users to build complex application; b) hiding the additional latencies that arise due to remote data access that sits across the network at the service provider sites; and c) most importantly, the challenge of data confidentiality – it was (and still is) well recognized that organizations view data to be amongst their most valuable assets. In the DAS model, the users data sits on the premises of the third-party service providers where it could be used / misused in ways that can no longer be controlled by data owners. The confidentiality challenge consisted of two distinct security problems: First, if the DAS model was to succeed, it was imperative that mechanisms be developed to empower service providers to provide sufficient security guarantees to the end-user. Appropriate firewall technologies, of course, help prevent malicious outsider attacks. Given the value of data to organizations, and the danger of litigations given data thefts, our goal became using encryption mechanisms to protect users confidentiality even if data at rest (in disks) gets stolen. Data management when data is stored encrypted in disks was less understood at the time (and to date has not been fully addressed). It led to a variety of challenges such as what granularity should the data be encrypted at (e.g., page level, record level, field level), what type of encryption technique should be used to encrypt data, how should key-management be performed, how should the pages/files be organized, how should indices be built upon encrypted data, what are the performance implications of encryption, what are the impacts on query optimization and processing, etc. Many of these challenges were addressed by us in subsequent papers [15, 10–12].

The second confidentiality challenge came from the issue whether end-users (data owners) will indeed trust service providers with their data (irrespective of

whether or not service providers follow best practices to prevent data from being stolen through malicious outsiders). There could be several reasons for distrust. The service provider model allows providers to be anywhere over the Internet, possibly outside international boundaries and hence outside the jurisdictional control of the legal systems accessible to data owners. Furthermore, as has been validated through numerous security surveys since then, insider attacks at the service providers are a major vulnerability in service-oriented models. Even if the service provider is a well-intentioned honest organization, a disgruntled or a malicious employee (of the service provider) could steal data for either personal gains or defaming a service provider.

This second challenge led us to explore a new approach to data management in the context of DAS model wherein the data is encrypted appropriately at the data owner site and is never visible in plaintext at the service provider. This way, even if there is the data stored at the service provider is made visible, data confidentiality is preserved since the data is encrypted. This was an ideal solution for the DAS model. However, the approach led us to the challenge of implementing SQL queries (i.e., relational operators such as selections, joins, etc.) over encrypted data representation. Cryptographic techniques that allow predicates to be evaluated over encrypted data representation (now known as searchable encryption) had not developed at the time [1, 2, 4, 21]. Even to date, such techniques are not efficient enough to be considered as a general solution that works well under most/all conditions. Instead, we devised an information theoretic approach which we called bucketization that allowed us to evaluate predicates though at the technique could admit false positives. This led us directly to partitioned computation of SQL queries collaboratively between the service provider and the client side wherein part of the predicate evaluation would be done at the service provider with some filtering at the client side to determine the true answer. This work on partitioned execution of SQL queries appeared in ACM SIGMOD 2002 conference wherein it was recently recognized through a SIGMOD Test-of-Time award in 2012.

1.2 Aggregations over Encrypted Data

While our SIGMOD paper addressed how predicate searches in SQL queries could be done collaboratively by appropriately partitioning the computation, it, nonetheless, had a gaping hole when it came to supporting even the simplest aggregation queries. To see this consider a simple aggregation query such as `SELECT sum(salary) from EMPLOYEE` that adds up the total salary of all the employees in an organization. While bucketization was devised to support predicate evaluations, and could hence be used to implement selections, joins, and through the algebra for partitioned computation, fairly complex SQL queries, it could not support aggregation efficiently. A query such as above would require the entire EMPLOYEE table (or at least the projection to the salary field to be transmitted across the network to the client side where it would be decrypted and then added. While bucketization provided us with a way to evaluate predicates, it did not provide us with a way to compute on the encrypted data directly. What

we needed was to incorporate homomorphic encryption techniques into the data representation to enable computing over encrypted data into the DAS model.

Homomorphic encryption techniques are a form of encryption that allow specific form of computations to be performed on cipher text such that the resulting output is the encrypted form of the true result of the computation. Thus, decrypting the resulting cipher text would yield the true answer to the query. Homomorphic encryption would permit us to compute the above aggregation query efficiently – if, for instance, the salary field was encrypted using such an encryption, we could add the salaries over the encrypted representation on the server, and then transmit the result to the client where the result could be decrypted. This would completely eliminate the need to transmit the whole relation to the client side bringing tremendous savings.

Homomorphic encryption can be classified as partially homomorphic that allows only one operation (i.e., either multiplication, or addition) to be performed over the encrypted domain or fully homomorphic that allows both the operations to be evaluated in cipher text. Examples of partially homomorphic techniques are unpadded RSA that allows multiplication, and Paillier cryptosystem that supports addition over encrypted representation. Given that aggregation queries could involve both multiplication and addition, we required a fully homomorphic encryption. The need and the power of the fully homomorphic cryptosystem has been long recognized. A secure solution to such a encryption method, however, remained an open problem in cryptography for over 30 years and has only recently been solved by the work of Gentry [6, 27, 5], which has recently led to a large number of innovations in homomorphic cryptosystems. To date, however, an efficient implementation of fully homomorphic system to be of practical value in general remains illusive to the best of our knowledge.

2 Aggregation Queries over Encrypted Data

Our focus and goal at the time, was not as much on improving cryptography, but rather on exploring how efficient query processing could be implemented in the DAS model if such a fully homomorphic crypto system were to be made available. To make progress, we chose the fully homomorphic cryptosystem, PH (Privacy Homomorphism), proposed originally in [26] even though it was known not to be fully secure. We give a background on PH as follows:

2.1 Background on Privacy Homomorphisms

Definition of PH: Assume $\mathcal{A}$ is the domain of unencrypted values, $\mathcal{E}_k$ an encryption function using key k, and $\mathcal{D}_k$ the corresponding decryption function, i.e., $\forall a \in \mathcal{A}$, $\mathcal{D}_k(\mathcal{E}_k(a)) = a$. Let $\tilde{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $\tilde{\beta} = \{\beta_1, \beta_2, \ldots, \beta_n\}$ be two (related) function families. The functions in $\tilde{\alpha}$ are defined on the domain $\mathcal{A}$ and the functions on $\tilde{\beta}$ are defined on the domain of encrypted values of $\mathcal{A}$. $(\mathcal{E}_k, \mathcal{D}_k, \tilde{\alpha}, \tilde{\beta})$ is defined as a privacy homomorphism if $\mathcal{D}_k(\beta_i(\mathcal{E}_k(a_1), \mathcal{E}_k(a_2), \ldots, \mathcal{E}_k(a_m))) = \alpha_i(a_1, a_2, \ldots, a_m) : 1 \leqslant i \leqslant n$. Informally, $(\mathcal{E}_k, \mathcal{D}_k, \tilde{\alpha}, \tilde{\beta})$ is a

privacy homomorphism on domain $\mathcal{A}$, if the result of the application of function α_i on values may be obtained by decrypting the result of β_i applied to the encrypted form of the same values.

Given the above general definition of PH, we next describe a specific homomorphism proposed in [26] that we will use in the remainder of the paper. We illustrate how the PH can be used to compute basic arithmetic operators through an example.

• The key, $k = (p, q)$, where p and q are prime numbers, is chosen by the client who owns the data.
• $n = p \cdot q$, p and q are needed for encryption/decryption and are hidden from the server. n is revealed to the server. The difficulty of factorization forms the basis of encryption.
• $\mathcal{E}_k(a) = (a \bmod p, a \bmod q)$, where $a \in \mathbb{Z}_n$. We will refer to these two components as the *p component* and *q component*, respectively.
• $\mathcal{D}_k(d_1, d_2) = d_1 q q^{-1} + d_2 p p^{-1} \pmod{n}$, where $d_1 = a \pmod{p}, d_2 = a \pmod{q}$, and q^{-1} is such that $qq^{-1} = 1 \pmod{p}$ and p^{-1} is such that $pp^{-1} = 1 \pmod{q}$. (1)
• $\tilde{\alpha} = \{+_n, -_n, \times_n\}$, that is addition, subtraction, and multiplication in mod n.
• $\tilde{\beta} = \{+, -, \times\}$, where operations are performed componentwise.

Example: Let $p = 5, q = 7$. Hence, $n = pq = 35, k = (5, 7)$. Assume that the client wants to **add** a_1 and a_2, where $a_1 = 5, a_2 = 6$. $\mathcal{E}(a_1) = (0, 5), \mathcal{E}(a_2) = (1, 6)$ (previously computed) are stored on the server. The server is instructed to compute $\mathcal{E}(a_1) + \mathcal{E}(a_2)$ componentwise (i.e., without decrypting the data). The computation $\mathcal{E}(a_1)+\mathcal{E}(a_2) = (0+1, 5+6) = (1, 11)$. The result, $(1, 11)$ is returned to the client. The client decrypts $(1, 11)$ using the function $(d_1 q q^{-1} + d_2 p p^{-1}) \pmod{n} = (1 \cdot 7 \cdot 3 + 11 \cdot 5 \cdot 3) \pmod{35} = 186 \pmod{35}$, which evaluates to 11, the sum of 5 and 6.[1] The scheme extends to **multiplication** and **subtraction**.

2.2 Extensions to PH

Notice that PH could directly be used to support aggregation in SQL queries since it is fully homomorphic. We were, however, faced with a few challenges listed below which became the topic of our technical contribution in this DAS-FAA 2004 paper. The full discussion can be found in [7].

Division Operation: The PH we use does not support division. SQL aggregation, however, requires the handling of division. As an example, consider the SQL clause `SUM (expense / currency_ratio)` on a table where expenses are listed in different currencies. Let us consider the first two terms of this summation; $\frac{a_1}{c_1} + \frac{a_2}{c_2}$. The numerator and denominator of the sum are $a_1 c_2 + a_2 c_1$ and $c_1 c_2$, respectively. Both only involve addition and multiplication, which are supported by the PH. The inability of the PH to handle division is compensated in query processing. The numerator and the denominator for the division are

[1] n is selected in such a way that results always fall in $[0, n)$; for this example $[0, 35)$.

computed separately and returned to the client for final computation. Then, the client decrypts both and performs the division.

Floating Point Number Representation: As defined, our PH is defined over the integer domain [26]. SQL has a float or real number data type. Our PH needs to be extended to handle real number arithmetic. Our suggestion is to treat real number as fractions. Arithmetic of fractions may be carried out using the technique used for division operation. Numerators and denominators are computed separately by the server on encrypted data and both sent to the client for final decryption and division. Consider the following example for floating point arithmetic.

Example 1. Let $p = 11, q = 13$. Hence, $n = pq = 143, k = (11, 13)$. Let us assume that the user wants to compute $x = a_1 + a_2$ for $a_1 = 1.8, a_2 = 0.5$. Hence, $\mathcal{E}(a_1) = ((7, 5), (10, 10)), \mathcal{E}(a_2) = ((5, 5), (10, 10))$.

$\mathcal{E}(a_1)+\mathcal{E}(a_2) = \frac{\mathcal{E}(a_1^n)}{\mathcal{E}(a_1^d)}+\frac{\mathcal{E}(a_2^n)}{\mathcal{E}(a_2^d)} = \frac{\mathcal{E}(a_1^n)+\mathcal{E}(a_2^n)}{\mathcal{E}(a_1^d)} = \frac{(7,5)+(5,5)}{(10,10)} = \frac{(12,10)}{(10,10)} \pmod{143}$

This result is sent to the client, and the client performs decryption as follows: $a_1 + a_2 = \frac{12 \cdot 13 \cdot 6 + 66 \cdot 11 \cdot 6}{10 \cdot 13 \cdot 6 + 10 \cdot 11 \cdot 6} = \frac{23}{10} \pmod{143} = 2.3$

Handling Negative Numbers: Negative numbers can be dealt by offsetting the range of numbers. To see the need for this, recall that arithmetic is defined on modulo n in PH. For example, the numbers 33 and -2 are indistinguishable when we represent them in modulo 35. That is, 33 (mod 35) ≡ -2 (mod 35).

Let the maximum value that can be represented in the computer system be $v_{max} > 0$ and the minimum value be $v_{min} = -v_{max}$. Then the possible value range would be $[v_{min}, v_{max}]$. We first map this range into another range, which is $[0, 2v_{max}]$. In this new range, any value $x < v_{max}$ is interpreted as a negative number relative to value of v_{max}. By defining that, an actual value x is mapped to a shifted

After these definitions, arithmetic operations should be mapped accordingly as well. For example, addition is mapped as $x + y = x' + y' - (v_{max} - v_{min})$ and similarly subtraction is mapped as $x - y = x' - y' + (v_{max} - v_{min})$. It is obvious that the server should have the encrypted value for $(v_{max} - v_{min})$, which will be computed with same encryption scheme and stored once. To illustrate the problem we stated above and the mapping scheme, consider the following example.

2.3 Security Extension to the PH

In this section we show how we addressed a security exposure, test for equality by the server, in the given PH scheme.

Preventing Test for Equality: Picking n such that $n > v_{max} - v_{min}$ (as we did above) enables the server to test for equality. Say x and y are two numbers encrypted as (x_p, x_q) and (y_p, y_q), respectively. Let $z = x * y$, which implies in the encrypted domain, $(z_p, z_q) = (x_p, x_q) * (y_p, y_q)$. The server could start adding (x_p, x_q) to itself every time checking the equality between the sum and

(z_p, z_q). When the equality is satisfied, the server learns the unencrypted value of y. Thus, $(z_p, z_q) = \underbrace{(x_p, x_q) + (x_p, x_q) + \ldots + (x_p, x_q)}_{y \ times}$.

We plug this exposure by adding random noise to the encrypted value. We encrypt an original value x as follows; $\mathcal{E}(x) = (x \pmod p) + R(x) \cdot p, x \pmod q + R(x) \cdot q)$, where $R(x)$ is a pseudorandom number generator with seed x. $R(x)$ value is generated during every insertion/update by the client. This prevents equality testing for the server and the server cannot remove the noise without knowing p and q. The noise is automatically removed at the client upon decryption. In the presence of noise, the following decryption function should be used in place of equation (1): $\mathcal{D}_k(d_1, d_2) = (d_1 \bmod p)qq^{-1} + (d_2 \bmod q)pp^{-1} \pmod n$ This equation is true because noise had been added in multiples of p for the first and in multiples of q in the second term. The modulo of each (mod p) and (mod q) term removes the added noise.

Another benefit of introducing the noise is that p and q components are no longer stored in modulo p and q, respectively. It makes it additionally difficult for the server to guess their values.

2.4 Query Processing over Encrypted Data

While PH provided us with a solution to implement the simple aggregation query about adding salaries of employees we discussed above, we were next faced with a challenge of computing aggregation queries that contain predicates. Most aggregation SQL queries are not as simple as the one used above to motivate the need for a fully homomorphic encryption approach. Consider, for instance, now a slightly modified query such as `SELECT sum(salary) FROM employee WHERE age > 45 and age < 50`. The bucketization not only selected terms which were part of the answer, but also false positives (which were eliminated on the client side). But this causes a problem since we cannot compute the answer at the server for aggregation. The basic idea we came across, which led us to the DASFAA paper, was that in bucketization we could treat buckets as sure or not sure. If we distinguish between them, we can compute the aggregation over sure things on the client side. Since most aggregations can be computed in a progressive manner, we could compute part of the aggregation on the server and transfer the rest when we were unsure to client side, decrypt and compute the answer on the client side. The strategy would not just work, but would also tune well by controlling the maximum size of each buckets and ensuring that number of unsure buckets given queries is limited. We give an overview of the process as follows:

Given a query Q, our problem is to decompose the query to an appropriate query Q^S on the encrypted relations R^S such that results of Q^S can be filtered at the client in order to compute the results of Q. Ideally, we would like Q^S to perform bulk of the work of processing Q. The effectiveness of the decomposition depends upon the specifics of the conditions involved in Q and on the server side representation R^S of the relations involved. Consider, for example, a query to retrieve sum of salaries of employee in $did = 40$. If did is a field-level encrypted field, the server can exactly identify records that satisfy the condition

by utilizing the equality between the client-supplied values and the encrypted values stored on the server. In such a case, aggregation can be fully performed on the *salary* attribute of the selected tuples exploiting the PH representation. If, on the other hand, the condition were more complex, (e.g., `did > 35 AND did < 40`), such a query will be mapped to the server side by mapping the *did* to the corresponding partitions associated with the *did* field that cover the range of values from 35 to 40. Since the tuples satisfying the server side query may be a superset of the actual answer, aggregation cannot be completely performed at the server. Our strategy is to separate the qualified records into those that certainly satisfy the query conditions, and those that may satisfy it - the former can be aggregated at the server, while the latter will need to be transmitted to the client, which on decrypting, can filter out those that do not, and aggregate the rest. The strategy suggests a natural partitioning of the server side query Q^S into two queries Q_c^S and Q_m^S as follows:

• **Certain Query (Q_c^S):** That selects tuples that *certainly* qualify the conditions associated with Q. Results of Q_c^S can be aggregated at the server.
• **Maybe Query (Q_m^S):** That selects *etuple*s corresponding to records that *may* qualify the conditions of Q but it cannot be determined for sure without decrypting. The client decrypts these *etuple*s, and then selects the ones that actually qualify and performs the rest of the query processing.

To finalize the computation, the client combines results from these queries to reach the actual answers. We next discuss how a client side query Q is translated into the two server side representations Q_c^S and Q_m^S.

3 Follow-up Work

The work described above on database-as-a-service was part of a successful collaboration between IBM and UCI that culminated in a Ph.D. dissertation of Hakan Hacıgümüş. Our follow up work, motivated and influenced in a large part by our initial papers on DAS has explored numerous challenges that arose when we started developing the DAS model. We discuss some of these additional challenges we explored subsequently below.

First, our approach to partitioned computation was based on supporting secure indexing tags by applying bucketization (a general form of data partitioning), which prevents the service provider from learning exact values but still allows it to check if the record satisfies the query predicate. Bucketization provided a natural sliding scale confidentiality that allowed us to explore a tradeoff between security and performance – e.g., if we map all data values to the same bucket, the approach provides perfect security (since adversary cannot distinguish between any data) but it incurs heavy overhead since now the service provider cannot prune records based on query predicates. Likewise, if we map each distinct value has a different bucket, the adversary gains significant knowledge about the data, but can also perform potentially perfect pruning in the context of query processing. Our subsequent work explored the natural tradeoff between security and performance that results in using bucketization as an

underlying technique for supporting SQL queries [13, 9, 14]. Another angle we explored the DAS model was towards exploiting additional benefits that the DAS model intrinsically provides. First amongst these is that outsourcing naturally creates the ability to seamlessly share data with others. Data sharing has, and to a large degree still remains a significant challenge – organizations often develop mutual (pairwise) data sharing protocols that implement their sharing policies. Individuals often share data with each other using diverse mechanisms such as emails, posting on publicly accessible websites, or physically sharing sharing data using memory sticks etc. DAS can significantly reduce the burden of data sharing, if a user could, besides outsourcing its data / query processing, could also outsource the task of data sharing to the database service. Such a solution would not just alleviate the responsibility of organizations to share the data, but it will also result result in improved performance and availability since the data (to be shared) already resides on the service providers. To a large degree, the technology has already adopted such an approach at least at the individual levels with the emergence and proliferation of services such as dropbox and google drive. Our work, however, explored sharing in the context when service providers were untrusted (and hence data was appropriately encrypted). Another benefit that outsourcing provides, particularly, in the context of personal data management, is that it empowers users to access data from anywhere, anytime, and from any device. This is particularly useful for data such as Web history, bookmarks, account information, passwords, etc .that a user may wish to access remotely using a mobile device or using different machines from environments that are not necessarily trusted. We explored mechanisms wherein trusted computation can be performed using the outsourced data model through the help of a small-footprint trusted proxy [18, 17, 16, 20, 19].

Since the time we explored the DAS model, the computing field has witnessed a major shift in computing paradigm – fueled by the advances in virtualization and high-speed networking, cloud computing model is emerging that can roughly be summarized as X as a service , where X can be virtualized infrastructure (e.g., computing and/or storage), a platform (e.g., OS, programming language execution environment, databases, web servers, etc.), software applications (e.g., Google apps), a service or a test environment, etc. Much like what motivated the industrial trend towards software-as a service (and also our work on DAS), the primary motivator of cloud computing is the utility computing model (aka pay-as-you go model) where users get billed for the computers, storage, or any resources based on their usage with no up-front costs purchasing the hardware/software or of managing the IT infrastructure. The cloud provides the illusion of limitless resources that one can tap into in times of need, limited only the the amount one is willing to spend on renting the resources. Our work on database as a service naturally fits the cloud model but the cloud offers many new challenges and opportunities that we did not originally explore as part of our work on DAS.

First, unlike our assumption in DAS, where the resources were assumed to be very limited on the client side, in the cloud setting organizations may actually possess significant resources that meets majority of their storage and computational

needs. For instance, in the cloud setting data may only be partially outsourced, e.g., only non-sensitive part of the data may be kept on the cloud. Also, it may be only at peak query loads that the computation needs to be offloaded to the cloud. This has implications from the security perspective since much of the processing involving sensitive data could be performed at the private side. In DAS, since the goal was to fully outsource the data and computation, the focus of the solutions was on devising mechanism to compute on the encrypted representation (even though such techniques may incur significant overhead). In contrast, in the cloud environments, since local machines may have significant computational capabilities, solutions that incur limited amount of data exposure of sensitive data (possibly at a significant performance gain) become attractive. Second, while our DAS work primarily dealt with database query workload, in a cloud setting, we may be interested in more general computation mechanisms (i.e. not only database workloads). For instance, map-reduce (MR) frameworks are used widely for large-scale data analysis in the cloud. We may, thus, be interested in secure execution of MR jobs in public clouds.

Another challenge arises from the potential autonomy of the cloud service providers. It is unlikely that autonomous providers will likely implement new security protocols and algorithms (specially given significant overheads associated with adding security and the restrictive nature of cryptographic security for a large number of practical purposes). For instance, it is difficult to imagine Google making changes to the underlying storage models, data access protocols and interfaces used by its application services (such as Google Drive, Picasa, etc.) such that users can store/search/process data in encrypted form. This calls for a new, robust and more flexible approach to implement privacy and confidentiality of data in cloud-based applications.

Our current research related to DAS is exploring the above challenges in the context of cloud computing[2]. In particular, we have explored a risk-aware data processing framework for cloud computing that, instead of focusing on techniques to eliminate possibility of attacks, provides mechanisms to limit / control the exposure risks. Different ways to steer data through the public and private machines and the way data is represented when it is on public machines exhibit different levels of risks and expose a tradeoff between exposure risks and system specific quality metrics that measure the effectiveness of a cloud based solution. Given such a tradeoff, the goal of the risk aware computing changes from purely attempting to maximize the application specific metrics to that of achieving a balance between performance and sensitive data disclosure risks. We have explored such a risk based approach for various contexts – HIVE and SparQL queries over relational data /RDF data as well as lookup queries over key value stores [25, 24, 22, 23, 3].

Another avenue of our work on cloud computing addresses the issue of autonomy of service providers by exploring an middleware-based approach for security where end-clients connect to cloud services through a security layer entitled

[2] This work is part of the Radicle Project (`http://radicle.ics.uci.edu`) at UCI which is funded through in part by the NSF Grant CNS 1118127.

CloudProtect. CloduProtect empowers users with control over their data that may store in existing cloud-based applications such as Box, Google drive, Picasa, Google Calendar, etc. It is implemented as a intermediary that intercepts http requests made by the clients, transforms the request to suitably encrypt/decrypt the data based on the users confidentiality policies before forwarding the request to the service provider. Encrypted representation of data may interfere with the users experience with the service - while requests such as CRUD operations (i.e., create, read, update, delete) as well as possibly search can be performed on encrypted representation, operations such as Google translate requires data to be in plaintext. CloudProtect keeps track of how data is stored on the server and based on the operation request may launch an exception protocol wherein encrypted data may be brought back to the client, decrypted and restored at the server before the service request is submitted to the server. Through the exception protocol, CloudProtect provides continued seamless availability of Web services. Furthermore, CloudProect supports mechanisms to adapt the data representation at the service providers to strike a balance between the risk of data exposure and the service usability (i.e., the number of times an exception protocol is invoked to meet users request).

Hakan Hacıgümüş followed up with the research work in the larger context of data management systems in the cloud. In that context, we worked on numerous challenges that need to be overcome before the database systems can be successfully integrated with the cloud delivery platforms. Some of the research areas include, elastic transactional database systems, large scale, flexible Big Data Analytics systems in the cloud, workload and resource management in cloud databases, data-rich mobile applications in the cloud. Many of the areas are explored in the CloudDB project at the NEC Laboratories of America [8].

References

1. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535–552. Springer, Heidelberg (2007)
2. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
3. Canim, M., Kantarcioglu, M., Hore, B., Mehrotra, S.: Building disclosure risk aware query optimizers for relational databases. PVLDB 3(1) (2010)
4. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proc. of ACM CCS (2006)
5. Gentry, C.: A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University (2009)
6. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proc. of STOC (2009)
7. Hacıgümüş, H.: Privacy in Database-as-a-Service Model. Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine (2003)
8. Hacıgümüş, H.: NEC Labs Data Management Research. SIGMOD Record 40(3) (2011)

9. Hacıgümüş, H., Hore, B., Iyer, B.R., Mehrotra, S.: Search on encrypted data. In: Secure Data Management in Decentralized Systems, pp. 383–425 (2007)
10. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Query Optimization in Encrypted Database Systems. In: Zhou, L.-z., Ooi, B.-C., Meng, X. (eds.) DASFAA 2005. LNCS, vol. 3453, pp. 43–55. Springer, Heidelberg (2005)
11. Hacıgümüş, H., Mehrotra, S.: Performance-Conscious Key Management in Encrypted Databases. In: Proc. of DBSec (2004)
12. Hacıgümüş, H., Mehrotra, S.: Efficient Key Updates in Encrypted Database Systems. In: Proc. of DBSec (2005)
13. Hore, B., Mehrotra, S., Hacıgümüş: Managing and querying encrypted data. In: Handbook of Database Security, pp. 163–190 (2008)
14. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proc. of VLDB (2004)
15. Iyer, B., Mehrotra, S., Mykletun, E., Tsudik, G., Wu, Y.: A Framework for Efficient Storage Security in RDBMS. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 147–164. Springer, Heidelberg (2004)
16. Jammalamadaka, R.C., Gamboni, R., Mehrotra, S., Seamons, K.E., Venkatasubramanian, N.: gvault: A gmail based cryptographic network file system. In: Proc. of DBSec (2007)
17. Jammalamadaka, R.C., Gamboni, R., Mehrotra, S., Seamons, K.E., Venkatasubramanian, N.: idataguard: middleware providing a secure network drive interface to untrusted internet data storage. In: Proc. of EDBT (2008)
18. Jammalamadaka, R.C., Gamboni, R., Mehrotra, S., Seamons, K.E., Venkatasubramanian, N.: A middleware approach for outsourcing data securely. Computers & Security 32 (2013)
19. Jammalamadaka, R.C., Mehrotra, S., Venkatasubramanian, N.: Pvault: a client server system providing mobile access to personal data. In: Proc. of StorageSS (2005)
20. Jammalamadaka, R.C., van der Horst, T.W., Mehrotra, S., Seamons, K.E., Venkatasubramanian, N.: Delegate: A proxy based architecture for secure website access from an untrusted machine. In: Proc. of ACSAC (2006)
21. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM Conference on Computer and Communications Security (2012)
22. Khadilkar, V., Oktay, K.Y., Kantarcioglu, M., Mehrotra, S.: Secure data processing over hybrid clouds. IEEE Data Eng. Bull. 35(4) (2012)
23. Oktay, K.Y., Khadilkar, V., Hore, B., Kantarcioglu, M., Mehrotra, S., Thuraisingham, B.M.: Risk-aware workload distribution in hybrid clouds. In: IEEE CLOUD (2012)
24. Oktay, K.Y., Khadilkar, V., Kantarcioglu, M., Mehrotra, S.: Risk aware approach to data confidentiality in cloud computing. In: Bagchi, A., Ray, I. (eds.) ICISS 2013. LNCS, vol. 8303, pp. 27–42. Springer, Heidelberg (2013)
25. Pattuk, E., Kantarcioglu, M., Khadilkar, V., Ulusoy, H., Mehrotra, S.: Bigsecret: A secure data management framework for key-value stores. In: Proc. of IEEE CLOUD (2013)
26. Rivest, R.L., Adleman, L.M., Dertouzos, M.: On Data Banks and Privacy Homomorphisms. In: Foundations of Secure Computation (1978)
27. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. IACR Cryptology ePrint Archive (2009)