Secure Database-as-a-Service with Cipherbase

Arvind Arasu
Microsoft Research
arvinda@microsoft.com

Spyros Blanas
University of Wisconsin, Madison
blanas@cs.wisc.edu

Ken Eguro
Microsoft Research
eguro@microsoft.com

Manas Joglekar
Stanford University
manas@cs.stanford.edu

Raghav Kaushik
Microsoft Research
skaushi@microsoft.com

Donald Kossmann
ETH Zurich
kossmann@inf.ethz.ch

Ravi Ramamurthy
Microsoft Research
ravirama@microsoft.com

Prasang Upadhyaya
University of Washington
prasang@cs.washington.edu

Ramarathnam Venkatesan
Microsoft Research
venkie@microsoft.com

ABSTRACT

Data confidentiality is one of the main concerns for users of public cloud services. The key problem is protecting sensitive data from being accessed by cloud administrators who have root privileges and can remotely inspect the memory and disk contents of the cloud servers. While encryption is the basic mechanism that can leveraged to provide data confidentiality, providing an efficient database-as-a-service that can run on encrypted data raises several interesting challenges. In this demonstration we outline the functionality of Cipherbase — a full fledged SQL database system that supports the full generality of a database system while providing high data confidentiality. Cipherbase has a novel architecture that tightly integrates custom-designed trusted hardware for performing operations on encrypted data securely such that an administrator cannot get access to any plaintext corresponding to sensitive data.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

Keywords

Security; Encryption; Trusted Hardware; Privacy

1. MOTIVATION

There has been significant interest in providing database-as-a-service [1, 8]. One of the most important concerns in the adoption of such services is security; in particular, data confidentiality. In fact, an organization may not even trust its own employees who operate a private cloud. As an example of a confidentiality breach, recently database administrators of several Swiss banks sold customer information to German and French tax authorities [13].

The problem only gets worse for the case of public clouds where an organization may trust a cloud provider to properly operate provisioned services, but may not trust the employees of the cloud provider to keep its data confidential. In this work, we assume we need to protect data against "honest-but-curious" adversaries, in particular cloud administrators who have root privileges in the servers and thus can inspect the contents of main memory and disks on these machines. Database administrators need these access privileges to do their job such as creating indexes, repartitioning the data, applying security patches to the machines, etc. However, these administrators do not need to see and interpret the data stored in the databases of those machines.

Encryption at rest [9, 11] is the principle used in mainstream database products such as Oracle and Microsoft SQL Server. Users (or database administrators) can specify that certain data (tables or partitions) be stored encrypted (e.g., using AES) on disk. In these systems, the disks are assumed to be untrusted while other system components (main memory, CPU) are assumed to be trusted (for instance, encryption keys can be stored in main memory). Thus, the data is decrypted in main memory to process queries and updates. While these systems protect data against media theft or from attackers with access to the storage system, they do not protect data against attacks from administrators with super-user privileges and thus are not suitable for the infrastructure to support a "secure" database-as-a-service.

A simple way to guarantee confidentiality is to treat the cloud database as an encrypted blob store and perform all query processing in a trusted client (which is the only component with access to the encryption keys). Since the cloud server never needs to decrypt data, such an architecture can guarantee the confidentiality of data. Although this approach addresses the threat of administrator attacks, in a database context, this approach has the following important limitations: 1) shipping encrypted data (e.g., full tables) to off-cloud machines may be prohibitively expensive for certain database workloads 2) as the amount of query processing done in the client increases, the benefits of using the cloud in the first place are significantly diminished.

In order to enable an efficient database-as-a-service, we need the ability to operate on encrypted data in the cloud. While generic homomorphic encryption techniques that enable arbitrary computation on encrypted data [7] are not yet practical, there are encryption techniques that enable executing different database operations directly on the encrypted data. For instance, if two columns are encrypted using deterministic encryption, the join can be evaluated by evaluating the join on the corresponding encrypted

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD'13, June 22–27, 2013, New York, New York, USA.
Copyright 2013 ACM 978-1-4503-2037-5/13/06 ...$15.00.

columns. CryptDB [12] is a recent system that exploits such partial homomorphic properties of various encryption schemes to support a subset of SQL queries. However, this technique is limited in that it cannot handle generic and ad-hoc SQL queries. For instance, there is no known partial homomorphic encryption scheme that can handle even simple aggregates such as $price * (1.0 - discount)$. Thus, any query that computes such aggregates over a table needs to ship the entire table to a trusted proxy in order to decrypt the data to evaluate the aggregate.

Cipherbase [3, 2] is an extension of Microsoft SQL Server, specifically designed to help organizations to protect their data against database administrators that have root access privileges on the database server. Cipherbase provides the same features as traditional database systems (e.g., support for full SQL, transactions, and recovery). Cipherbase features a novel hardware / software co-design that leverages customized hardware (based on FPGAs [10]) in which the keys can be stored in a tamper-proof way in order to enable computations on encrypted data *in the cloud* in a secure fashion.

Cipherbase's architecture guarantees *orthogonal security* because it allows organizations to develop their applications and set their data security goals relatively independently of any performance, scalability, or cost considerations. For instance, consider a join query involving two columns encrypted using strong encryption (e.g., AES-CBC mode in which the same plaintext value will get encrypted to multiple values). Since the encryption is not deterministic, the join cannot be directly evaluated on the encrypted values. While prior systems like [12] cannot evaluate such a join — Cipherbase can leverage the secure hardware to evaluate such queries. Our tightly coupled design is also fundamentally different from the loosely coupled design of TrustedDB [4], that combines an IBM secure co-processor (SCP) and a commodity server (TrustedDB runs a lightweight SQLite database on SCP and a more feature-rich MySQL database on the commodity server). The fine-grained integration in Cipherbase enables a smaller footprint for the trusted hardware module as well as novel optimizations (for a more detailed comparison with Trusted DB see [3]). The secure hardware essentially serves as a secure proxy for the client in the cloud and as a result mitigates the limitations we discussed for the pure blob store approach. Thus, Cipherbase is particularly suited to be used as the infrastructure for a "secure" database-as-a-service.

In Section 2, we present a brief technical overview of Cipherbase and in Section 3 we outline the demo scenario and how users can interact with our system during the demonstration.

2. TECHNICAL OVERVIEW

In this section, we provide a brief overview of the Cipherbase system. For a more detailed overview of the architecture, refer to [3]. The goal of the Cipherbase project is to develop a novel cloud computing platform that allows organizations to leverage the advantages of cloud computing and at the same time achieve data confidentiality. The Cipherbase system provides the same features as traditional database systems (e.g., support for full SQL, transactions, and recovery). In terms of confidentiality, the Cipherbase system supports various levels of encryption (from no encryption to strong encryption) and different end-to-end security settings so that the right level of confidentiality can be selected for all data.

2.1 Architecture

Figure 1 gives an overview of the Cipherbase system. Applications (or end users) at client machines issue SQL queries and updates using, e.g., ODBC, embedded SQL, or a console just like

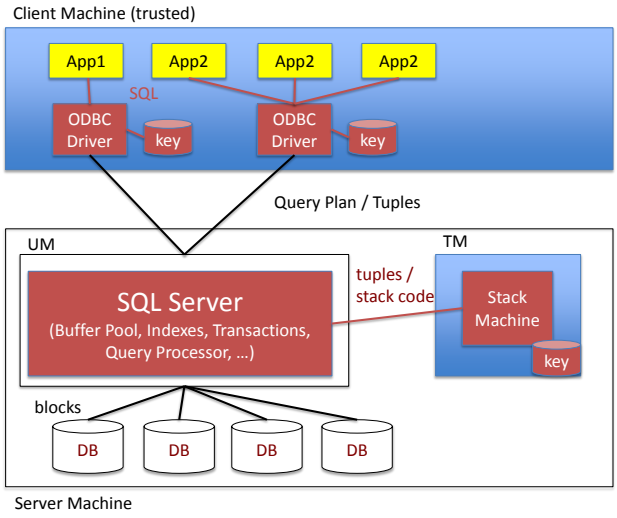

Figure 1: Cipherbase Architecture

in any other database system. These SQL statements are handled by the database driver at the client (e.g., an ODBC driver) and then processed by the server; again, just like in any other database system. Furthermore, Cipherbase has the same components as traditional database systems such as a storage manager (buffer pool, indexes, etc.) that reads/writes data in blocks from/to disks, a transaction manager (concurrency control, write-ahead logging, etc.), and a query processor (optimizer, runtime system, etc.).

In addition to the database server denoted as a UM (for Untrusted Module), Cipherbase integrates a secure database processor which is represented with a blue background and denoted as a TM (for Trusted Module) in Figure 1. The TM is used as submodule for core operations over encrypted data. The TM uses an FPGA [6] and is placed inside the UM, connected via a high-speed 8x PCI Express bus. The TM runs a stack machine that evaluates expressions compiled into stack code as part of the query plan. Thus, interpretation of the query plan by the Cipherbase runtime system involves shipping (encrypted) tuples from the UM to the TM, decrypting, processing, and re-encrypting these tuples in the TM, and shipping the (encrypted) results back from the TM to the UM.

Again, security comes at a cost and the performance degrades if all data requires strong confidentiality. So, organizations should carefully specify the level of confidentiality for all data. For instance, public data such as country names should be marked as public and highly confidential data such as customer names should be marked as such. Cipherbase exploits these settings by optimizing queries and transactions taking the encryption and security of all data into account.

2.2 Encryption Policy

Applications can set their data security goals by specifying an *encryption policy* which are additional annotations on the schema definitions. Figure 2 gives an example of a simple schema for patient and disease information. Every patient has a *name* and an *age*. A disease has a *name* and a *description*. Furthermore, the database keeps track of which patient had which disease. All patient information is considered to be confidential. The patient names are strictly confidential so that the schema of Figure 2 specifies that they be encrypted using AES. As *names* are unique in the *Patient* table, deterministic encryption (ECB mode) is okay. *Patient.age*

is less critical; in particular, if the age cannot be associated to a name. As a result, *Patient.age* can be encrypted using a weaker, order-preserving technique such as ROP [5]. Disease information is public and the diagnosis information is again highly confidential because it belongs to patients. As shown in Figure 2, any kind of integrity constraint can be implemented, independent of the encryption technique. Just as for regular query processing, however, the choice of the encryption technique impacts the performance of maintaining integrity constraints.

```
create table Patient (
name      :  VARCHAR(50) AES_ECB primary key,
age       :  VARCHAR(50) ROP check >= 0);

create table Disease (
name      :  VARCHAR(50) primary key,
descr     :  VARCHAR(256));

create table Diagnosis (
patient   :  VARCHAR(50) AES_CBC
               references Patient,
disease   :  VARCHAR(50) AES_ECB
               references Disease,
date      :  DATE check not null);
```

Figure 2: Specifying Confidentiality Needs

In addition to setting policies for protecting data-at-rest, Cipherbase also offers application developers the ability to specify annotations for securing data-in-motion. For more details, refer to [2].

2.3 Query Processing

The UM and the TM constitute an asymmetric distributed system: the TM is secure but since it is based on specialized hardware is resource limited, while the UM is (potentially) insecure but powerful. This asymmetry holds independent of whether we use FPGAs or other alternatives, and argues for a design that minimizes the TM footprint. Our overall design involves revisiting each module of the database system and identifying core primitives that need to operate on encrypted data and factoring these out to be implemented in TM (more details on the exact set of primitives that we identify and the rationale for picking these are presented in [3]). Here, we note that, interestingly, a small class of primitives involving encryption, decryption, and expression evaluation suffices to support query processing, concurrency control, and other database functionality.

The Cipherbase server runs an extended database system (SQL Server) with some components modified to make round-trips to the TM for operations over encrypted data. For example, index lookup invokes a "find position" primitive for each index page traversed to identify the next page to visit. The main advantage of this design is that we are able to leverage the relatively powerful UM for operations that do not depend on encryption, even when data is strongly encrypted. For example, the hash join operator uses the TM for computing hashes and checking equality. The rest of hash join logic including the code for memory management, writing hash buckets to disk, and reloading them all run in the UM. The second advantage is software engineering: by attaching hooks to a small number of places in the SQL Server code and routing data to the TM, we are able to get rich functionality of an industrial strength database system. The third advantage, as discussed earlier, is that by keeping the TM functionality simple, we are able to leverage the power of FPGAs.

Given the tight coupling between the UM and the TM, there are optimizations in Cipherbase to manage the communication latency between TM and UM; recall that they are connected by a PCIexpress bus. Optimizations include careful design of primitives to minimize round trips and various kinds of inter- and intra-query batching of TM invocations. In addition, the query optimizer has extensions in the form of additional transformation rules in order to optimize round trips to the TM. Some example additional rules include: 1) *Move to UM*: If data is public or encrypted in a partially homomorphic way that matches the operation (e.g., order-preserving encryption), then the operation can be executed wholly by UM 2) *Merge*: If there is sufficient memory available in the TM, then two operators can be executed in the TM without shipping tuples back and forth from the TM to the UM.

In terms of security guarantees, we assume an adversary can monitor all operations in the UM, the traffic between the UM and TM but cannot monitor any computation in the TM. Again, the query plan specifies when and which tuples are shipped to the TM and which functions (i.e., stack code) the TM applies to the tuples. In order to decrypt and re-encrypt tuples, the application-specific secret key is known to the TM; it is not available and visible anywhere in the UM. More precisely, each TM has its own secret key which is burnt into hardware and stores the secret keys of applications in an encrypted format (using its own secret key) on the disks of the UM. Moreover, any traffic to the TM only involves encrypted data. As a result,Cipherbase can prevent an administrator who can inspect the main memory and disk contents from obtaining the plaintext corresponding to sensitive data.

3. DEMO SCENARIO

In the demonstration we will show a prototype of Cipherbase in which we have modified Microsoft SQL Server and interfaced it with a custom designed FPGA to implement the trusted module. The system will also be instrumented from the perspective of an adversary — an administrator can get access to the memory and disk contents at any instant. Users can interact with Cipherbase using the standard tools of SQL Server such as Management Studio (a snippet is shown in Figure 3) in order to issue queries.

Demo visitors can interact with the demo under multiple roles.

- **System User**: In order to evaluate the orthogonality of Cipherbase, users can define encryption policies (or view multiple predefined policies in which the number of encrypted columns is varied) and examine how query plans change as a result - especially in terms of computations performed in the TM (e.g., the secure join operator in Figure 3) and in general evaluate the performance overheads for different encryption policies for any particular query.

- **Untrusted Administrator**: In order to evaluate the security offered by Cipherbase, we will first demonstrate how only data-at-rest encryption is insufficient and how by using carefully designed memory dump utilities, an untrusted administrator can learn about sensitive data. We can then demonstrate how such an attack can be avoided in the Cipherbase architecture.

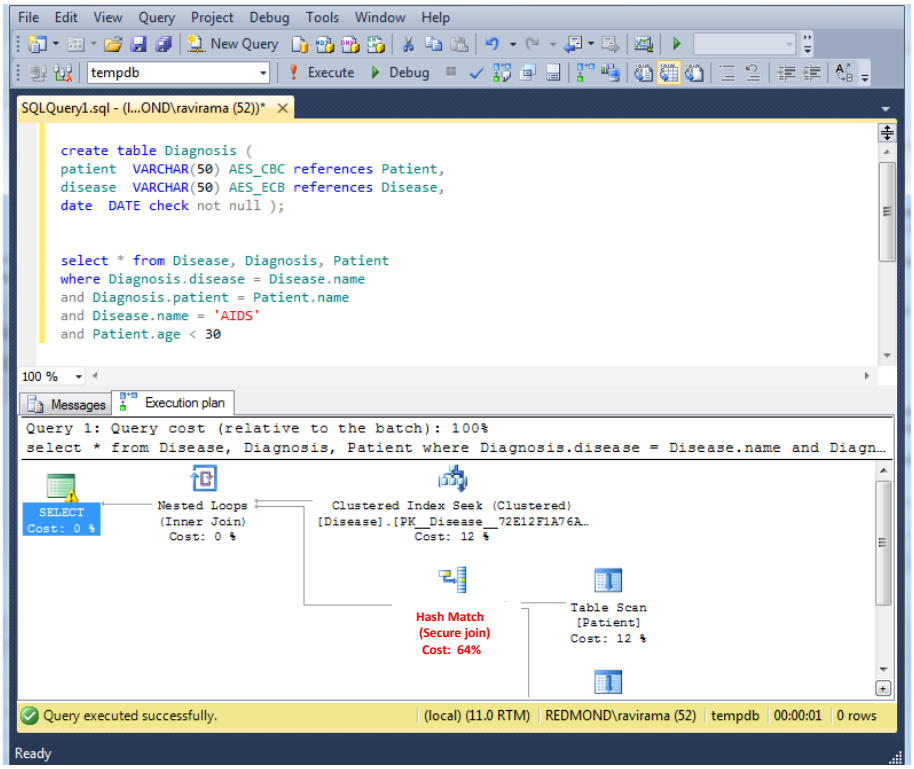

Figure 3: Query Plan using TM

4. REFERENCES

[1] Amazon Corporation. Amazon Relational Database Service. http://aws.amazon.com/rds/.

[2] A. Arasu et al. Engineering security and performance with cipherbase. In *Data Engineering Bulletin Vol 35.(4)*, 2012.

[3] A. Arasu et al. Orthogonal security with cipherbase. In *CIDR*, 2013.

[4] S. Bajaj and R. Sion. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *SIGMOD*, 2011.

[5] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT '09*, 2009.

[6] K. Eguro and R. Venkatesan. FPGAs for trusted cloud computing. In *FPL*, 2012.

[7] C. Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3), 2010.

[8] Microsoft Corporation. SQL Azure. http://www.windowsazure.com/en-us/home/features/sql-azure/.

[9] Microsoft Corporation. SQL Server Encryption. `http://technet.microsoft.com/`.

[10] R. Müller, J. Teubner, and G. Alonso. Data processing on fpgas. volume 2, 2009.

[11] Oracle Corporation. Transparent Data Encryption. `http://www.oracle.com/`.

[12] R. A. Popa, C. M. S. Redfield, N. Zeldovich, et al. Cryptdb: protecting confidentiality with encrypted query processing. In *SOSP*, pages 85–100, 2011.

[13] Germany Tackles Tax Evasion. Wall Street Journal, Feb 7 2010.