A SECURE COPROCESSOR FOR DATABASE APPLICATIONS

Arvind Arasu[*], *Ken Eguro*[*], *Raghav Kaushik*[*], *Donald Kossmann*[†],
Ravi Ramamurthy[*], *Ramarathnam Venkatesan*[*‡]

Microsoft Research
[*]Redmond, WA USA and [‡]Bangalore, India
email: {arvinda, eguro, skaushi,
ravirama, venkie}@microsoft.com

[†]ETH-Zurich
Zurich, Switzerland
email: donaldk@ethz.ch

ABSTRACT

The scalability and availability of cloud computing makes it an ideal platform for many database applications. However, it is challenging to secure sensitive client information in a practical and rigorous manner against both external attackers and curious cloud administrators. In this paper, we describe a novel secure FPGA-based query coprocessor and discuss how it can be tightly integrated with a commercial database system such as SQL Server. This combination, called Cipherbase, leverages efficient division of labor – using a conventional untrusted cloud server to handle mundane database operations while sensitive data is segregated and processed in trusted hardware to ensure confidentiality. We examine the architectural design issues that affect the achievable performance of the system and report initial results demonstrating the effectiveness for real-world cloud database applications.

1. INTRODUCTION

Several vendors such as Amazon RDS and SQL Azure offer cloud database-as-a-service. The low startup cost, scalability, and high-availability of these systems encourage organizations to migrate their database applications to the cloud. However many applications have sensitive information (e.g. employee payroll salaries, Social Security numbers, etc.) which needs to be secured in order to be placed in the cloud.

While most commercial database products support encryption [9][10], these solutions are insufficient from a security standpoint. This is because they only encrypt the database file while on disk. Although this protects against an adversary that physically steals a hard drive, during normal operation encrypted data is decrypted and stored as plaintext in main memory (not to mention the cryptographic keys to perform the decryption). This makes data accessible to a cloud administrator or hacker that can gain root privileges. A recent study [12] reflects this concern, showing that a primary deterrent towards moving applications to the cloud is precisely this problem of securing sensitive data from the service provider.

The only feasible solution to this issue is to encrypt sensitive data before storing it in the cloud. The problem now is how to safely perform database operations on encrypted data without dramatically affecting the system's usability or performance. As we will discuss, existing solutions for operating on encrypted data have limitations in terms of generality, performance, cost, or security. These issues negate the advantages of migrating database applications like online transaction processing (OLTP) to the cloud.

In this paper, we outline a new architecture for servicing cloud database applications efficiently. A key component in this system is an FPGA-based secure database coprocessor. When tightly integrated with a commercial database system, this secure coprocessor enables the execution of SQL queries on encrypted data entirely in the cloud. This occurs without handling plaintext data or cryptographic keys in a conventional server where a user with root privileges could gain access. This system, Cipherbase, provides customers with strong security guarantees and a flexible, cost-effective way of ensuring the efficient migration of database applications to the cloud.

2. SECURING CLOUD DATABASE PROCESSING

In this section, we discuss different alternatives by which we can secure cloud database processing. To make the discussion concrete, consider a simple banking application which maintains the accounts of different customers. The sensitive attributes that need to be protected include the customer ID (Social Security number) and their account balance. The queries that need to be executed on this database include computing the new account balance after a deposit or withdrawal and calculating the new account balance to reflect earned interest.

Secure Servers: As shown in Fig. 1, one approach is to secure an entire server in-cloud where a database system (DBMS) can run using plaintext. Similar to what is provided by Amazon GovCloud or high-end private cloud solutions, these servers are physically and logically isolated (e.g. in cages with cameras and guards, and placed on separate secured networks). In general, this approach is very costly to implement due to the unique physical

environment the service provider has to maintain and due to the management issues this isolation creates (e.g. complicated failover and load balancing).

Perhaps more seriously, though, it is very difficult to make rigorous security guarantees for these isolated servers. For example, many common security problems stem from the fact that these servers are built from general-purpose processors. General-purpose processors contain a single, physically unified memory space for both program and data. This makes them highly adaptable, but also opens the door for exploits such as buffer overruns and rootkits that can defeat memory protection mechanisms.

Thus, the security of these systems can only be guaranteed if the entire software stack can be proven to be bug-free. However, as shown in [11], formally verifying even a simple OS kernel with 8,700 lines of C and 600 lines of assembly is highly non-trivial. A modern cloud server built on a complex hypervisor and running full-featured virtual machines simply has too large a surface area to be formally verified in practice.

(Partial) Homomorphic Encryption: In contrast to secure servers where data is store unencrypted and operated upon in a trusted location, homomorphic encryption techniques enable computation directly on ciphertext. From a security perspective, this is ideal since the cloud database only stores ciphertext while still able to compute arbitrary SQL queries on the data. Since no keys or plaintext ever exist on cloud machines, the operations can run on untrusted machines in full view of would-be attackers.

However, to date, no computationally tractable fully homomorphic encryption technique [5] has been found. There are only efficient partial homomorphic encryption (PHE) techniques that can perform specific operations. For instance, the Paillier cryptosystem [7] can perform addition on ciphertext using public key algorithms. Similarly, deterministic encryption (e.g. AES-EBC mode) consistently produces the same ciphertext when presented with the same plaintext. This allows equality evaluation and, by extension, database operations such as grouping and joins.

CryptDB [8] is a recent system that supports a subset of SQL queries using such PHE techniques. However, since different PHE approaches use different underlying cryptographic principles, they are incompatible with one another. In the case of our simple banking application, there is no efficient PHE technique that can support both additions and multiplications on ciphertext. Thus, existing PHE techniques do not provide a sufficiently generic solution appropriate for arbitrary cloud database processing.

Trusted Client: The previous two approaches perform secure computation entirely in-cloud. Trusted client based techniques rely on a combination of untrusted cloud and trusted customer-owned resources. This is a currently popular solution for many cloud "success" stories [13].

Here, data is stored using conventional encryption techniques in an untrusted (indicated with red in Fig. 2) cloud DBMS. The cloud DBMS connects to a trusted (indicated with blue) component in a customer-end machine which has the encryption key. Since the cloud database is untrusted and does not have access to the key, any computation that requires manipulation of ciphertext must be performed in the trusted client.

This results in distributed query evaluation between the trusted client and the untrusted cloud DBMS. Consider the task of computing earned interest. The system must transmit the encrypted balance to the trusted client, decrypt the value, update it, and re-encrypt before sending the new value back to the cloud DBMS. Here, the overhead of transmitting data back and forth between the client and cloud server easily dwarfs the time actually spent computing. In fact, for many common queries this approach may result in shipping a large fraction of the database to the trusted client (e.g. calculating the sum of all accounts). Thus, this approach can incur a non-trivial performance penalty as well as a cost penalty (the cost to maintain client-side servers and the cost of bandwidth consumed shuttling data between the cloud and client). This issue seriously reduces the benefits of migrating the database to the cloud.

Trusted Hardware: As we have seen, existing secure cloud database approaches all have limitations in terms of generality, performance, cost, or security. At the same time though, the general concept of distributed query evaluation cleanly divides work among trust and untrusted compute resources. Trusted compute is needed to handle operations on sensitive data, but untrusted resources can be used for other database operations – e.g. storage, retrieval, and logging services, computations that run on non-sensitive plaintext data, or computations that do not need to directly manipulate ciphertext. The only real issue with the trusted

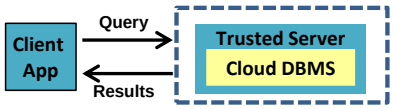

Fig. 1. Trusted Server Approach

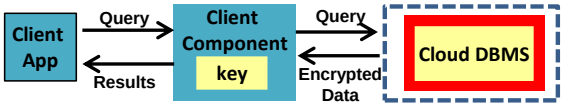

Fig. 2. Trusted Client Approach

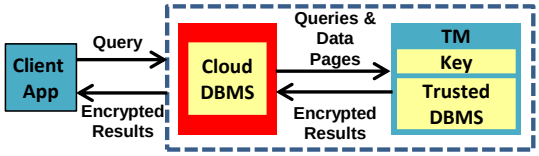

Fig. 3. Loosely-Coupled Architecture

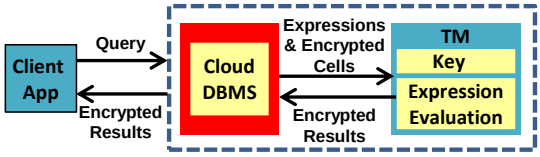

Fig. 4. Tightly-Coupled Architecture

client model is that the trusted compute resources are physically so far removed from the cloud itself. Thus, we could successfully leverage the advantages of the cloud (without exotic servers or encryption techniques) if we had a secure in-cloud location where we could store encryption keys, decrypt sensitive values, compute database operations, and re-encrypt the results.

Towards this end, using dedicated hardware as a secure co-processing platform is promising in terms of providing security for a fixed set of operations. Simply implementing these operations in a separate device helps protects the data, but beyond this, as discussed in [4], purpose-built circuits present a much smaller attack surface as compared to secure server solutions. Formal verification techniques are more tenable and the fundamental nature of custom hardware addresses many of the vulnerabilities of general-purpose processor-based systems. For example, memory address spaces can be physically disjoint since we simply do not need the same degree of freedom as in a general-purpose processor. Similarly, many of the complexities (and potential for bugs) associated with software-based systems revolve around multi-tasking an inherently sequential processor. Dedicated hardware, on the other hand, can natively support multi-tasking with completely independent circuits.

In terms of the cloud, FPGAs are a particularly attractive platform to provide dedicated hardware. While offering many of the previously mentioned advantages of dedicated circuits, they also offer a large degree of flexibility. This is important because a cloud provider can purchase and install a single chip that can be quickly repurposed to support a wide variety of applications. This allows the cloud to adapt these resources to meet current and future demands. When not running secure database operations, they could be used for other secure cloud computing applications [4], or play a more traditional role as high-performance computational accelerators.

3. SECURE DATABASE COPROCESSOR DESIGN

In this section, we describe the design of a secure database co-processor. As discussed earlier, this trusted compute resource augments an untrusted cloud DBMS by performing secure operations on sensitive data.

3.1. System Integration

As shown in Fig. 3 and Fig. 4, there are two basic architectures for distributing query evaluation between the trusted and untrusted computing platforms: loosely and tightly coupled. In both options, the database encryption key is only present on client machines and on the trusted co-processor in the cloud (denoted as TM). The fundamental difference between these two architectures is in way that query execution is divided between the cloud DBMS and the TM. Specifically, they differ in the granularity of operations.

In a loosely-coupled architecture, the TM contains a full DBMS and is responsible for executing entire queries or sub-queries. Queries or sub-queries that do not involve sensitive information are executed on the untrusted cloud DBMS, while queries that involve sensitive information are executed by the TM. The fact that the TM in a loosely-coupled architecture contains a complete DBMS creates both advantages and disadvantages.

Perhaps the most important benefit of a loosely-coupled solution is that it is relatively easy to build. Since the TM is a self-contained DBMS, the client only needs to change the database schema to reflect which fields are encrypted. The queries themselves also need to be re-written to reflect the split nature of the computation, but this can utilize existing database features for distributed query processing, such as user-defined functions and remote stored procedures. This means that loosely-coupled systems can use largely off-the-shelf software and hardware components.

At the same time, the loosely-coupled TM must be fairly complex to provide a full secure DBMS. For example, in our taxonomy, TrustedDB [3] is a loosely-coupled architecture. TrustedDB utilizes an IBM 4764 secure cryptographic processor running Linux and SQLite. The complexity of this type of TM creates two problems.

First, as mentioned in Section 2, the need for a full OS and DBMS has security implications due to the system complexity. Second, as a secure, fully self-contained system, the TM naturally has limited computational and storage resources. The high functionality requirements of a loosely-coupled architecture seriously stretch the TM's capabilities. For example, the onboard memory in the 4764 is very limited. Thus, to allow the TM to work on realistically-sized databases, TrustedDB stores all database pages in the primary untrusted DBMS (sensitive values are encrypted, so this does not create a security issue). As the TM processes queries, it requests database pages from the untrusted system.

In general, this arrangement leads to an inefficient use of both the precious secure computational power in the TM and the bandwidth between the TM and the outside world. This limits the achievable performance in high-throughput cloud applications. For example, aside from the essential cryptographic and data manipulation operations in queries that must run in the TM, the TM in a loosely-coupled architecture also parses queries, manages database pages, manages requests for database pages (only some of the data within a given page may actually be used), manages change logs, etc.. None of these operations manipulate ciphertext values and, thus, could be run in an untrusted system.

This brings us to the tightly-coupled architecture used in our work, Cipherbase. Here, the TM only contains expression evaluation functionality. Expression evaluation is the lowest-level database computational abstraction and

is the part of SQL Server responsible for actually manipulating basic data types. This includes computations such as comparison, arithmetic and SQL intrinsic functions (MIN, MAX, etc.). SQL Server compiles all queries into a series of these simple expressions. In a conventional SQL Server installation, a logically separate data storage engine fetches and feeds the appropriate database pages to a simple expression interpreter that performs the necessary computation. The Cipherbase tightly-coupled architecture piggybacks on this existing computational model, augmenting conventional expression evaluation on plaintext in the untrusted DBMS with secure expression evaluation on ciphertext in the TM.

This type of tightly-coupled architecture has two specific advantages. First, the TM is comparatively simple, only implementing a small set of processing primitives. This greatly reduces the complexity of the TM and allows us to build a dedicated platform that can be provably secure. Second, the TM is highly efficient, both in terms of computational resources and bandwidth. During runtime, the TM only processes those operations that explicitly require access to the real values contained in encrypted fields. That is, all data management and staging tasks are offloaded to the comparatively cheap and plentiful computational resources in the untrusted DBMS. Furthermore, only values that the TM explicitly needs to read are transferred by the untrusted DBMS, conserving bandwidth to the outside world.

At the same time, to implement a tightly-coupled architecture also requires non-trivial changes to the existing untrusted DBMS codebase. Although for brevity we will not discuss the necessary modifications in detail here, briefly, the complication stems from the fact that the distribution of operations now occurs at a very fine-grained level. This redefines the basic operating procedures of expression evaluation, going beyond the capabilities of existing external hooks (e.g. user-defined functions) to achieve appropriate integration and acceptable performance. That said, considering the very high efficiency and security that is achievable, we have elected to implement a tightly-coupled architecture.

3.2. Securing the FPGA

Regardless as to how the system divides work between the trusted and untrusted components, client confidence in the system as a whole hinges upon the security of the TM. Beyond the fact that the Cipherbase TM is a purpose-built circuit and, thus, naturally more resistant to hacking than pure software, it is important that 1) the FPGA is loaded with a known and trusted bitstream and 2) the device can be uniquely identified by remote clients.

The use of FPGAs as a trusted cloud computing platform has been discussed in prior work [4]. Here, we rely on many of the same basic concepts: a trusted third-party authority, standard FPGA bitstream protection hardware/techniques, and standard public-key infrastructure and key exchange mechanisms.

As shown in Fig. 5, this process begins with a third-party authority, trusted by both the cloud operator and customers. The trusted authority generates an AES bitstream encryption key unique to a particular FPGA (or equivalent group of FPGAs). This key is then uploaded into the device, privately and before it is deployed in the cloud. The trusted authority then creates a unique RSA public/private key pair for each FPGA and inserts the private key into the bitstream representing the TM. This new binary is then encrypted and signed with the corresponding FPGA's bitstream encryption key. This protected bitstream is then transferred to non-volatile memory on the appropriate FPGA board inside the cloud. Finally, the public identity of the FPGA is published using standard public-key infrastructure (PKI).

This process ensures security in several ways. First, both the cloud operator and clients trust the signing authority to validate the logic inside the TM bitstream (i.e. that it does exactly what is intended – nothing more and nothing less). Second, since the bitstream is signed, we can guarantee that this binary will load onto the FPGA unaltered. Any alterations to the signed bitstream will be detected during startup and cause the FPGA to fail the loading process. Third, since the bitstream (and thus the private RSA key inside) is encrypted, the identity of the FPGA cannot be stolen. When a client looks up the public key of a particular device and initiates communication, they are certain that they are truly connecting with the intended trusted recipient.

Before moving on, note that in some scenarios, the trusted authority could be the cloud service provider itself. That is, customers may trust the cloud provider as an organization to build and install the TM, but may not carry this faith to individual cloud administrators.

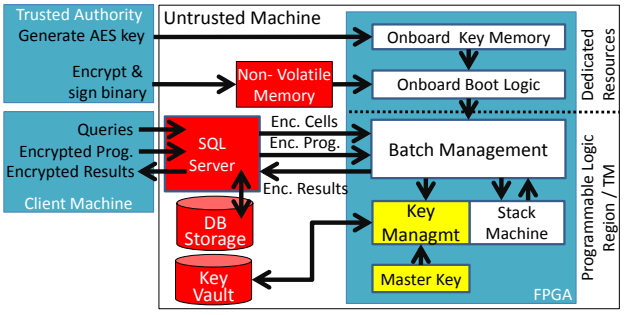

Fig. 5. Database coprocessor details

Table 1. SQL Operations and TM Primitives

SQL Operation (Plaintext)	Primitives in TM
SELECT …. WHERE **A = 5**	Dec(***A***) = Dec(**5**)
SELECT **A+B** WHERE …	Enc(Dec(***A***) + Dec(***B***))
SELECT … WHERE **T1.a = T2.b** …	Hash(Dec(***A***)),Hash(Dec(***B***)) Dec(***A***) = Dec(***B***)

3.3. Trusted Module Design

As discussed in Section 3.1, we only need to implement expression evaluation in a tightly-coupled secure coprocessor to support general-purpose query processing. In Table 1, we provide a few examples of basic expression evaluation operations in SQL. In the left column, we show a plaintext query and in the right column we show the corresponding primitives that would execute in the TM if the operation were performed on encrypted data. For instance, for a SELECT query with a predicate ($A = 5$) where column A is encrypted, the TM evaluates the comparison between the encrypted column and the corresponding encrypted constant. Similarly for computing ($A+B$), the TM decrypts both columns, adds them and then re-encrypts the result. Finally, for the join predicate between tables *T1* and *T2*, the query processor would match records in the two tables. The TM decrypts the values, hashes the join attributes, and then checks for equality. For a more detailed set of primitive examples, refer to [1].

SQL Server implements expression evaluation using a stack machine, compiling all tasks into database primitive programs that implement queries and sub-queries. Cipherbase leverages this existing computational abstraction. We built a stack machine in the FPGA-based TM that mirrors the "virtual machine" that traditionally runs in SQL Server. As a research prototype, our current hardware implementation does not support the complete suite of SQL stack operations, which number in the few hundreds. However, as we will see in Section 5, our implementation is sufficient to allow us to explore the potential (and potential pitfalls) of a full end-to-end secure database application for transactional workloads.

Fig. 5 depicts the different components of the complete Cipherbase system. Each Cipherbase server runs a modified SQL Server instance that has access to a TM. The first client (the database owner) sets up a schema for their database to indicate the format, including which fields are to be protected with encryption. This schema is signed and then installed into SQL Server and distributed among users of the database. Other clients can then connect to this SQL Server instance to upload data.

In the simplest execution model, this data is encrypted by clients before being loaded into SQL Server. In this case, the database owner defines a fixed cryptographic key for particular encrypted fields when defining the schema. This key is shared to other users and uploaded to the TM. Although the database owner and other clients can only communicate with the TM through the untrusted DBMS, clients can securely send keys to the TM by encrypting their keys with the TM's public key. These wrapped keys will then only be accessible to the TM, which can decrypt the client's keys with its private key. Notice that these wrapped keys are safe regardless as to how they are transmitted. Wrapped keys can even be cached by SQL Server in a key vault for later re-transmission to the TM.

At this point, the system is ready to service queries. To run queries, clients must first modify some of their normal SQL queries to reflect when they need manipulate encrypted fields. Although we will describe an example of this in more detail in the next section, specifically, these modified queries call out to stack programs that run on the TM. These programs are encrypted and signed so that they are protected in transit. When received by SQL Server, these programs are sent to the TM and cached for later reuse. When the untrusted DBMS executes queries or sub-queries over encrypted data, it packs the appropriate ciphertext together with a program ID, referring to which trusted program to execute on the data. This forms a work unit, ready for transmission to the TM. Multiple such work units are aggregated by the untrusted DBMS for batched transfer and execution.

Notice that the use of Cipherbase is largely transparent to clients (i.e. they do not have to modify their current applications to use Cipherbase). This is accomplished by installing a thin "shim" program on the client. This shim would be responsible for managing keys, encrypting/decrypting transmissions to and from the cloud, and converting queries to reflect the encrypted database schema.

Also notice that Cipherbase does not address the issue of correctness. For example, a malicious attacker could duplicate/withhold queries or insert random ciphertext. The issue of correctness is orthogonal to the primary focus of this paper (i.e. incorrect behavior is acceptable as long as no secrets are leaked). The problems of database validation and verification have been studied in the context of conventional databases and much of this work can be extended to a distributed system such as Cipherbase.

4. END-TO-END EXAMPLE

In this section we describe the operation of Cipherbase with an example query from the banking application discussed in Section 2. Consider the following plaintext SQL query:

```
UPDATE Accounts
SET AcctBal = AcctBal + :TransAmt
WHERE AcctID = :ID
```

This update takes two input parameters: *ID*, the account to be updated, and *TransAmt*, the amount of a transaction. Assume that the *Accounts* table has the schema (*AcctID*, *AcctBal*) associating each account with a balance.

With Cipherbase, we can specify the encryption for each attribute separately – see [1] and [2] for more details. For the purpose of this example, assume the table encrypts *AcctID* using deterministic encryption (e.g. AES-ECB mode) and *AcctBal* using stronger, non-deterministic encryption (e.g. AES-CBC mode). *TransAmt* arrives from the client encrypted. Cipherbase executes the update query with the following series of operations:

1) The untrusted DBMS looks up the *ID* record using an index (the *AcctID* column is encrypted using deterministic encryption which preserves equality, eliminating the need to access plaintext).
2) The untrusted DBMS fetches the encrypted *AcctBal* value for this row.
3) The untrusted DBMS sends *AcctBal*, *TransAmt* and the appropriate wrapped encryption key to the trusted module. The trusted module executes the following program, returning the result to the untrusted DBMS:

a) PUSH *AcctBal*	f) DECRYPT
b) PUSH *key*	g) ADD
c) DECRYPT	h) PUSH *key*
d) PUSH *TransAmt*	i) ENCRYPT
e) PUSH *key*	j) POP

4) The untrusted DBMS updates the table with the encrypted *AcctBal* result and writes a log record to persist the effect of this operation.

5. EXPERIMENTS AND RESULTS

We built two proof-of-concept Cipherbase platforms in which SQL Server communicates with an FPGA-based TM either via Gigabit Ethernet or x4 PCIe v2.0. Although PCIe is a faster, lower latency transport mechanism, an Ethernet solution would more easily allow multiple cloud DBMS machines to share a single TM.

The FPGA resource requirements for these two prototypes are shown in Table 2. Although our existing test platform TMs do not implement the full suite of SQL stack machine operations, the current resource utilization is very low – the core database processor uses less than 5% of the available V6LX240T. Thus, we believe that there is sufficient headroom to expand the functionality of the system in the future. The core database processor in both platforms is clocked at 125MHz.

We compare the performance of Cipherbase running on fully encrypted data against the performance of unmodified SQL Server running on plaintext data. All tests were performed on a Windows Server 2008 machine with dual 2.0GHz Intel Xeon E5-2650 processors (for a total of 32 logical cores) and 64GB of DDR3 RAM.

The goal of this evaluation is to understand the overhead of using encryption with the Cipherbase architecture. Our current system design and evaluation is focused on transactional workloads; such workloads are update intensive and each transaction (or query) touches a small number of records. In contrast, analytical workloads are read-oriented and each query typically touches a large number of records. Such workloads present new optimization opportunities and we plan to explore this space in future work.

TPC-A Benchmark: To evaluate Cipherbase, we use an industry standard TPC-A benchmark [6]. While simple, this benchmark is fairly representational of transactional workloads. Briefly, the benchmark models a bank with 10 branches. Each branch has 10 tellers and 100,000 customer accounts. Each account, teller, and branch has a corresponding balance field in the database. Every transaction in the benchmark performs a deposit or withdrawal from a random customer account. Each customer account update is then also reflected in the corresponding teller and branch balances.

For the encrypted version of the benchmark, we assume that all fields are encrypted as described in Section 4. Since the primary key fields (i.e. *AcctID*) are encrypted using deterministic encryption, indexing is performed outside the FPGA co-processor. All other fields are encrypted using strong non-deterministic encryption, and so the three balance updates required to process every transaction occur within the FPGA.

The primary evaluation metric for the benchmark is the number of transactions per second (TPS) the system is able to sustain. In our evaluation, a driver program running on the same machine as SQL Server issues customer transactions. The number of concurrent threads in the driver program represents the maximum number of simultaneous transactions that can be requested. This is a control parameter in the benchmark and for all our experiments we varied the number of driver threads from 1 to 500 (until the TPS reached a maximum).

TPC-A Relative Performance to Plaintext: Fig. 6 compares the *TPS* for the plaintext (*PT*) and two ciphertext (*CT-PCIe* and *CT-Ethernet*) database systems. In all our graphs, *TPS* values are normalized to the maximum throughput achieved for a given benchmark[1]. All numbers are warm numbers (i.e. the database is entirely cached in main memory and the system does not need to read values off of the hard drive). This is the most challenging scenario for our system, since the I/O overhead in many common workloads will mask performance shortcomings in the processing engine.

Fig. 6 shows that for this benchmark, the PCIe and Ethernet-based Cipherbase implementations achieve a peak throughput of 0.83*x* and 0.80*x* the maximum plaintext

Table 2. Resource Utilization (V6 LX240T).

	LUTs	***FF***	***BRAM***	***DSP***
Full System - Ethernet	9.1K (6.0%)	6.1K (2.0%)	67 (16.1%)	4 (0.5%)
Full System - PCIe	21.5K (14.3%)	20.7K (6.9%)	102 (24.5%)	4 (0.5%)
· Ethernet Infrastructure	*1.9K (1.3%)*	*1.1K (3.6%)*	*51 (12.2%)*	*0 (0.0%)*
· PCIe Infrastructure	*14.3K (9.5%)*	*15.7K (5.2%)*	*86 (20.7%)*	*0 (0.0%)*
· DB Proc.	*7.2K (4.8%)*	*5.0K (1.7%)*	*16 (3.8%)*	*4 (0.5%)*

[1] Due to a *DeWitt Clause*, we are unable to report any absolute performance numbers and must use normalized comparisons.

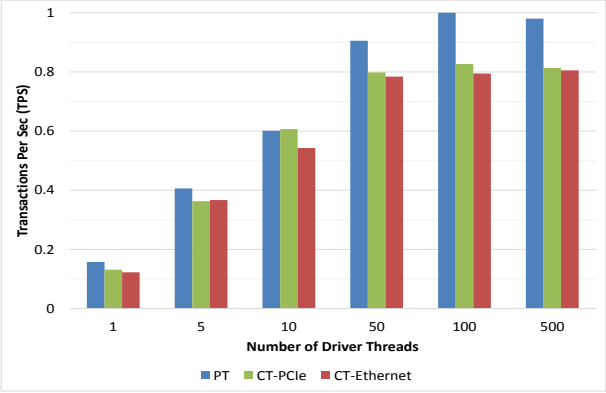

Fig. 6. Performance for original TPC-A, normalized to peak plaintext throughput

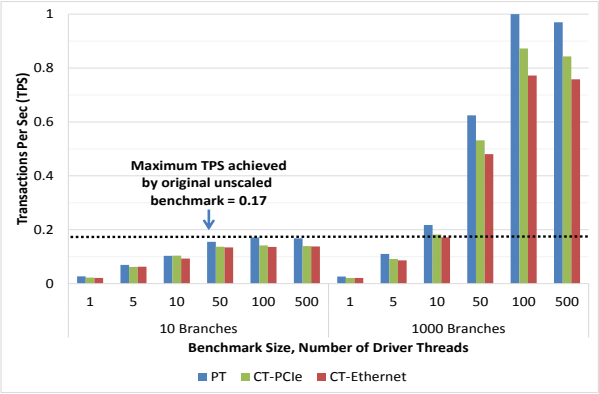

Fig. 7. Performance for original #branches=10 and scaled #branches =1000 TPC-A benchmark, normalized to peak plaintext 1000 branch throughput

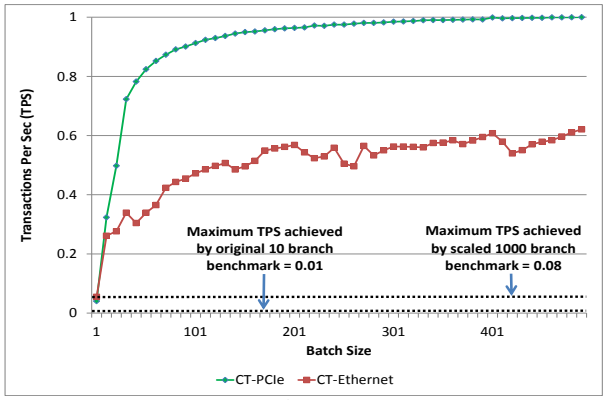

Fig. 8. TM processing performance with pre-packed data

database performance, respectively. Thus, the throughput degradation compared to plaintext processing is (1.0-0.83=17%) and (1.0-0.80=20%), respectively.

Although this is an acceptable penalty, especially given that the robust security of Cipherbase dramatically changes the fundamental functionality of the system, we would like to examine what issues contribute to this overhead.

There are two issues that penalize the performance of Cipherbase. First, Cipherbase simply handles more data than an insecure SQL instance. Rather than using a single 32-bit plaintext word to represent a balance, each encrypted field expands by a factor of four to a 128-bit AES block. In future work we plan to mitigate this data expansion.

A second issue that causes the measured Cipherbase overhead in this test is insufficient concurrent work. Each secure operation must call out to the external TM. Thus, the latency of this processing is naturally higher than if we remained in-processor. Although this added delay is relatively small (on the order of microseconds), it is incurred along a very fundamental part of the database's processing path. If we do not have a sufficient level of concurrency to amortize the latency to the TM, the host DBMS will be underutilized, waiting for data to return before processing can continue.

Although we expect the level of concurrency to be very high in most large-scale cloud applications, the original TPC-A benchmark has an unusually low degree of concurrency. Recall that there are only 10 bank branches and that each customer transaction also updates both the teller and branch balances. To maintain correct behavior in the face of simultaneous transaction requests, SQL Server must lock the respective customer account, teller or branch balance when it is being modified. This prevents new transactions for a particular value to proceed. Thus, regardless as to how many driver threads are present, on average no more than 30 operations can be serviced at a given time (10 accounts, 10 tellers, and all 10 branches).

Benchmark Scaling: To test the theory that the previously measured performance of Cipherbase is artificially low in the TPC-A benchmark due to limited effective concurrency, we repeated the previous experiment. Here, we scale the benchmark to contain 1000 branches (again, each with 10 tellers and 100,000 customer accounts). In this scaled benchmark, the system could theoretically processes up to 3000 operations concurrently (updates to 1000 customer accounts, 1000 tellers and all 1000 branches). As shown in Fig. 7, the increased level of concurrency dramatically improves performance. For example, the maximum throughput for even the conventional plaintext system increases by a factor of (1/0.17=5.9*x*), indicating that the system was indeed starved for parallel work and spinning on locked fields.

Cipherbase also benefits from the increased concurrency. The PCIe-based Cipherbase implementation now achieves a peak of 0.87*x* the maximum plaintext database performance. This reduces the overhead of this implementation to (1.0-0.87=13%). At the same time, the added concurrency accentuates the performance difference between the PCIe and Ethernet based systems. The Ethernet implementation achieves 0.77*x* the maximum plaintext performance, for a penalty of (1.0-0.77=23%).

Measuring TM Utilization: One last question we have regarding the performance of Cipherbase is the utilization of the FPGA itself. As mentioned earlier, the clock rate of the TM is very low compared to that of the primary

processor. Although there are many operation performed by the DBMS to fetch and stage data, log transactions, etc., the system could be computationally limited by the TM.

Thus, we developed a standalone executable independent of SQL Server to examine the load on the TM. This program produces a series of *N* TPC-A compliant queries. It then repeatedly sends this series of computations to the TM and receives the results. Since this system relies on pre-packed data with no operational load beyond sending and receiving data from the TM, this simulates the maximum computational load on the FPGA for this benchmark if the rest of the untrusted DBMS processing were infinitely fast.

As shown in Fig. 8, the maximum achievable throughput of hardware by itself is very high and the TM is dramatically underutilized when running in conjunction with SQL Server. The maximum measured *TPS* of the original live-running plaintext TPC-A benchmark is only (0.01/1.0=1%) of the maximum measured capabilities of the PCIe Cipherbase system and (0.01/0.62=1.6%) that of the Ethernet Cipherbase system. Similarly, the maximum measured *TPS* of the live-running plaintext scaled TPC-A benchmark is only (0.08/1.0=8%) of the capabilities of the PCIe Cipherbase system and (0.08/0.62 =13%) that of the Ethernet Cipherbase system.

6. CONCLUSIONS & FUTURE WORK

In this paper, we outline a new architecture for servicing database applications efficiently in the cloud. Our system leverages a novel secure database coprocessor that provides an important security guarantee – any sensitive data in the database will be decrypted only in a trusted module. We presented the design of an FPGA-based specialized database stack machine with a modest footprint, only needing to implement expression evaluation in the trusted processor.

We have integrated a prototype of this processor with a commercial DBMS and our preliminary performance evaluation on transactional workloads indicates that this architecture can provide performance very close to that of a conventional insecure system. Furthermore, our experiments shows that a single database installation is not capable of saturating the computational capabilities of the TM for a TPC-A style workload – there are simply too many non-processing oriented tasks required to stage and track the various operations. This finding has three implications.

First, this suggests that multiple database instances may be able to share a single TM for similar workloads (e.g. one FPGA per rack of conventional servers). This would amortize the cost of additional hardware among more customers or more work. Second, the system has considerable performance headroom. This is important for future work as we migrate to benchmarks with more complex secure processing needs. Third, this suggests that a tightly-coupled architecture is able to achieve better performance as compared to a loosely-coupled architecture. Cipherbase offloads non-processing oriented database operations to highly scalable cloud host machines. If these extraneous database operations needed to run on the comparatively scarce resources of the TM, the overall performance of the system would likely be lower.

Looking to the future, we are in the process of evaluating the performance of the system for other workloads, such as analytical queries. Beyond this, we are also exploring further optimizations to better integrate the trusted co-processor with the untrusted DBMS software stack.

7. REFERENCES

[1] A. Arasu, S. Blanas, K. Eguro, M. Joglekar, R. Kaushik, D. Kossman, R. Ramamurthy, P. Upadhyaya, R. Venkatesan. *Orthogonal Security with Cipherbase*. CIDR 2013.

[2] A. Arasu, S. Blanas, K. Eguro, M. Joglekar, R. Kaushik, D. Kossman, R. Ramamurthy, P. Upadhyaya, R. Venkatesan. Engineering Performance and Security with Cipherbase. IEEE Data Engineering Bulletin 35(4) 2012. 65-72.

[3] S.Bajaj, R.Sion. TrustedDB: A Hardware based Outsourced Database Engine. VLDB 2011 Vol. 4, 1359-62.

[4] K.Eguro, R.Venkatesan. FPGAs for Trusted Cloud Computing. FPL 2011. 63-70

[5] C. Gentry. A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University, 2009.

[6] J. Gray. The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann 1993.

[7] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. EUROCRYPT 1999. 223-38.

[8] R. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. SOSP 2011. 85-100.

[9] SQL Server Transparent Data Encryption. http://msdn.microsoft.com/en-us/library/bb934049.aspx

[10] Oracle Transparent Data Encryption. http://www.oracle.com/technetwork/database/options/advanced-security/

[11] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, S. Winwood "seL4: formal verification of an OS kernel" SOSP 2009. 207-20.

[12] An SME perspective on cloud computing. European Network and Information Security Agency, 2009.

[13] Amazon AWS Case Studies, http://aws.amazon.com/solutions/case-studies