Towards a simple modern design for VDB

Some VDB papers

Database Outsourcing with Hierarchical Authenticated Data Structures

Mohammad Etemad   Alptekin Küpçü
Crypto Group, Koç University, Istanbul, Turkey
{metemad, akupcu}@ku.edu.tr

vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases

Yupeng Zhang*, Daniel Genkin†,*, Jonathan Katz*, Dimitrios Papadopoulos‡,* and Charalampos Papamanthou*
*University of Maryland   †University of Pennsylvania   ‡Hong Kong University of Science and Technology
Email: {zhangyp,cpap}@umd.edu, danielg3@cis.upenn.edu, jkatz@cs.umd.edu, dipapado@cse.ust.hk

***Abstract*—Cloud database systems such as Amazon RDS or Google Cloud SQL enable the outsourcing of a large database to a server who then responds to SQL queries. A natural problem here is to efficiently verify the correctness of responses returned**

ever, they suffer from limited expressiveness, and in particular they cannot handle a wide range of SQL queries.

A. Our Results

Super-Efficient Verification of Dynamic Outsourced Databases*

Michael T. Goodrich[1], Roberto Tamassia[2], and Nikos Triandopoulos[3]

[1] Dept. of Computer Science, UC Irvine, USA, goodrich@ics.uci.edu
[2] Dept. of Computer Science, Brown University, USA, rt@cs.brown.edu
[3] Dept. of Computer Science, University of Aarhus, Denmark, nikos@daimi.au.dk

Abstract. We develop new algorithmic and cryptographic techniques for authenticating the results of queries over databases that are outsourced to an untrusted responder. We depart from previous approaches by considering *super-efficient* answer verification, where answers to queries are validated in time asymptotically less that the time spent to produce them and using lightweight cryptographic operations. We achieve this property by adopting the decoupling of query answering and answer verification in a way designed for queries related to range search. Our techniques allow for efficient updates of the database and protect against replay attacks performed by the responder. One such technique uses an

IntegriDB: Verifiable SQL for Outsourced Databases

Yupeng Zhang
ECE Dept. & UMIACS
University of Maryland
zhangyp@umd.edu

Jonathan Katz
CS Dept. & UMIACS
University of Maryland
jkatz@cs.umd.edu

Charalampos Papamanthou
ECE Dept. & UMIACS
University of Maryland
cpap@umd.edu

ABSTRACT

This paper presents INTEGRIDB, a system allowing a data owner to outsource storage of a database to an untrusted server, and then enable anyone to perform verifiable SQL queries over that database.

most prevalent applications running in the cloud today, and could be offered as a software layer on top of any SQL implementation.
Toward this end we design, build, and evaluate INTEGRIDB, a system that efficiently supports verifiability of a rich subset of SQL

Design Criteria—What we’d like

- Practical
  - Concrete efficiency
  - Conceptual simplicity
  - Development simplicity
    - (example benchmark: I should be able to program it in a few focused days)
  - The ”right” level of expressivity
    - It won’t need to be almighty
- Modern
  - should potentially use tricks from the last few years
    - E.g., (potentially) aggregations SnarkPack-style, subvector commitments, SNARKs in few localized places, potentially Caulk, etc.

Example DB

Source of table: IntegriDB paper

row_ID	student_ID	age	GPA	First_name	Type
1	10747	22	3.5	Bob	M.Sc
2	10715	24	3.3	Alice	B. Sc
3	10721	23	3.7	David	Ph.D
4	10781	21	3.0	Cathy	M.Sc

Table “Students”

student_ID	title	topic
10721	“Tarantino and critical theory”	cinema
10715	“An intro to homotopy theory”	math
10801	“Lars Von Trier and Dogma 95”	cinema

Table “FinalWriteups”

Simple query example

row_ID	student_ID	age	GPA	First_name
1	10747	22	3.5	Bob
2	10715	24	3.3	Alice
3	10721	23	3.7	David
4	10781	21	3.0	Cathy

Table Students

```
SELECT
* FROM Students WHERE
Age IN (21,22,23)
```

Simple JOIN query

row_ID	student_ID	age	GPA	First_name	Type
1	10747	22	3.5	Bob	M.Sc
2	10715	24	3.3	Alice	B. Sc
3	10721	23	3.7	David	Ph.D
4	10781	21	3.0	Cathy	M.Sc

Table “Students”

student_ID	title	topic
10721	“Tarantino and critical theory”	cinema
10715	“An intro to homotopy theory”	math
10801	“Lars Von Trier and Dogma 95”	cinema

Table “FinalWriteups” (FW)

SELECT
Students.**student_ID**,
Students.**GPA**,
B.**topic**
FROM Students INNER JOIN FW
ON Students.**student_ID** = FW.**student_ID**

Features of the construction

- + good with categorical data (MSc/BSc/PhD, small ranges)
- “Simple” queries
- Think: static databases for now
  - We can think of extending it later
  - NB: static DB == setting with observed preprocessing
- Constant size openings
- Main ingredients:
  - Vector commitments
  - Accumulators with some special properties
    - We’ll see them later

First Attempt: one VC per column

row_ID	student_ID	age	GPA	First_name
1	10747	22	3.5	Bob
2	10715	24	3.3	Alice
3	10721	23	3.7	David
4	10781	21	3.0	Cathy

When do we build this list of VCs?
Think at preprocessing time (again, for now it's a static DB, or a DB where all updates are observed and preprocessed honestly)

VC was not enough; let’s add something else

row_ID	student_ID	age	GPA	First_name
1	10747	22	3.5	Bob
2	10715	24	3.3	Alice
3	10721	23	3.7	David
4	10781	21	3.0	Cathy

Table Students

```
SELECT
* FROM Students WHERE
Age = 21
```

VC was not enough b/c we did not know if there were giving **all** rows satisfying a certain property
(VC can only tell you “this subset is in the col”, not necessarily the *maximal* subset)

Idea: Since we are preprocessing the DB, why not having a preprocessing that helps us a little bit with this?

Let’s add an “Oracle”

- Before we look at the Auth. Data Structure corresponding to this, let’s assume we have a magic oracle for the following
- Let’s then see if oracle + VC give us query responses that make sense

The Oracle **O**(T,C,v):
- Given table T, column C, value v
- Return { id: T.C[id] = v } // *id dependent on ctx (think: some key)*

student_ID	title	topic
10721	“Tarantino and critical theory”	cinema
10715	“An intro to homotopy theory”	math
10801	“Lars Von Trier and Dogma 95”	cinema

Table “FinalWriteups” (FW)

Example:
O(FW, topic, “cinema”) ->
{ 10721, 10801 }

Simple query with oracle + VC

student_ID	title	topic
10721	"Tarantino and critical theory"	cinema
10715	"An intro to homotopy theory"	math
10801	"Lars Von Trier and Dogma 95"	cinema

Table "FinalWriteups" (FW)

```
SELECT
* FROM FW
WHERE topic =”cinema”
```

Verifier:

- Receive results Rslt
- Asks **O**(FW, topic, "cinema") *// we implement this later as ADS; no worries*
- Receive response Idxs_oracle
- Receive VC proofs from prover for each col in Rslt (see below)
- For each column $Col in Rslt:
- - VC.Vfy(vc: VC_FW_$Col,
  prf: prf_$Col,
  vals: Rslt[$Col],
  Idxs: Idxs_oracle)

Recall:
O(FW, topic, "cinema") ->
{ 10721, 10801 }

Implementing the oracle

The Oracle **O**(T,C,v):

- Given table T, column C, value v
- Return S_v // NB: it's a set
- - *where S_v :=* { id: T.C[id] = v }

The Data Structure for **O**(T,C,v):

- Given table T, column C, have a "VC" such that
- VC = VC.Commit([v1 : Acc_v1, v2 : Acc_v2, …]) // it's more a KV map
- - *where Acc_v := Accum*({ id: T.C[id] = v })

Loose ends: size of parameters. A problem?

- What I presented so far has a “digest” linear in the total number of columns.
  - One VC per column + one “oracle ADS” per column
- Is that a problem?

How to do JOINs

Table "Students"

row_ID	student_ID	age	GPA	First_name	Type
1	10747	22	3.5	Bob	M.Sc
2	10715	24	3.3	Alice	B. Sc
3	10721	23	3.7	David	B.Sc
4	10781	21	3.0	Cathy	M.Sc

Table "FinalWriteups" (FW)

student_ID	title	topic
10721	"Tarantino and critical theory"	cinema
10715	"An intro to homotopy theory"	math
10801	"Lars Von Trier and Dogma 95"	cinema

Verifier:

- Receive results Rslt
- Asks **O**(Students, type, B.Sc)
- Asks **O**(FW, topic, *) // all topics
- Receive responses Idxs_oracle_Students, Idx_oracle_FW
- Check that Rslt[id] == Idxs_oracle_Students ∩ Idx_oracle_FW // new op!
- Receive VC proofs from prover for each table/col in Rslt (see below)
- For each table $T and each column $Col in Rslt:
- - VC.Vfy(vc: VC_$T_$Col,
  prf: prf_$T_$Col,
  vals: Rslt[$Col],
  Idxs: Idxs_oracle)

```
SELECT
S.Id, S.Name, topic FROM
S INNER JOIN FW
WHERE S.type = "B.Sc"
```

How to implement intersections?

- Check that Rslt[id] == Idxs_oracle_Students ∩ Idx_oracle_FW // new op!
- Recall this ”oracle” is implemented as accumulators
- We need accumulators supporting intersection
- KZG-based accumulators have this property!
  - KZG accum = Commit(characteristic polynomial of set)
  - Use divisibility for intersections

Other loose ends

- What I am looking at:
  - Sorting gadgets (useful on their own and for updates)
  - VCs are often homomorphic. Let’s exploit that!
    - Gadgets to prove:
      SELECT * FROM C1, C2 WHERE C1+2*C2 = 0

      SELECT * FROM C1, C2 WHERE C1+2*C2 > 0

-