Answering Aggregation Queries in a Secure System Model

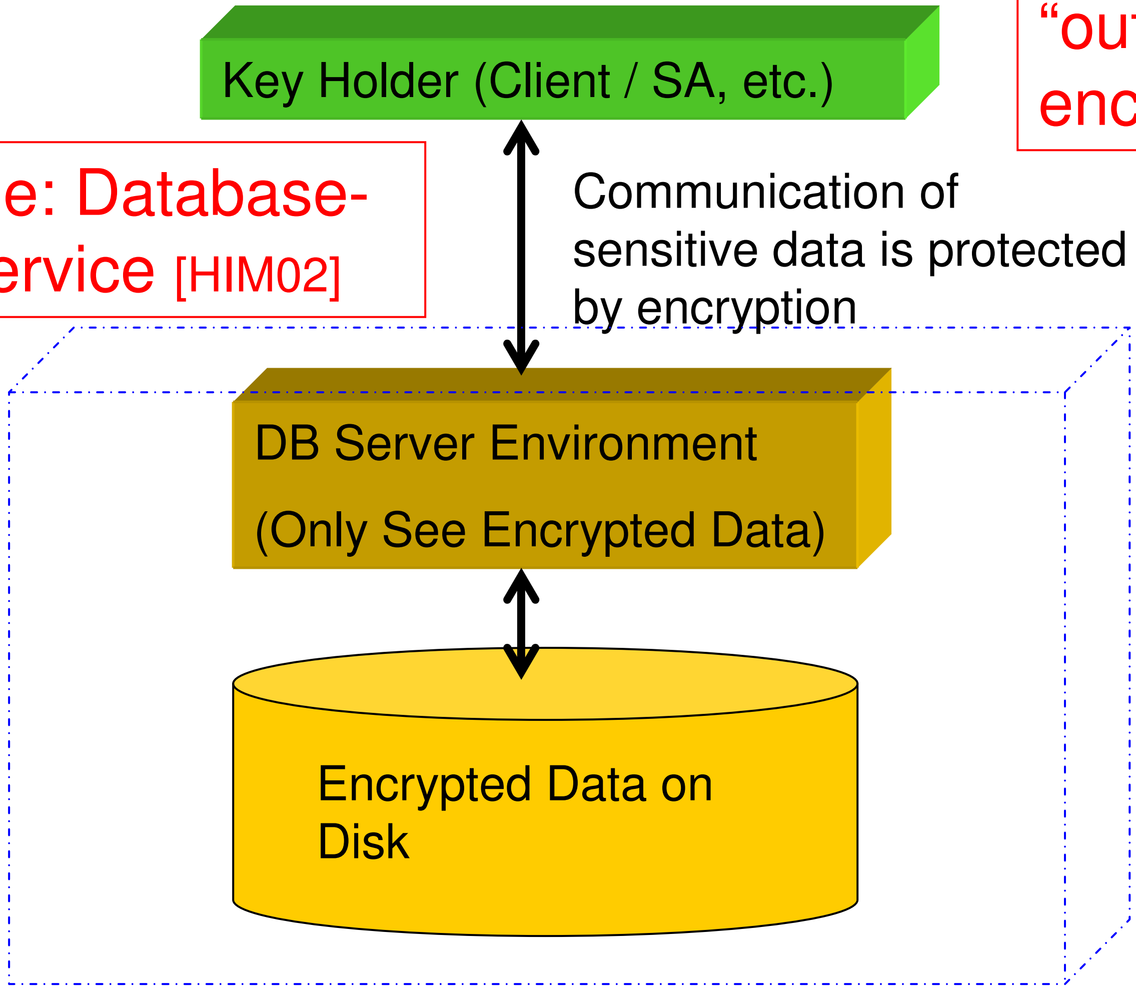
Tingjian Ge, Stan Zdonik
Brown University

BROWN

System Model

"outside-the-box" encryption model

Example: Database-As-a-Service [HIM02]



Separation of Duties

Server logs, config files, temp tables, memory are secure

Communication protected

Key distribution & management easy

Privacy & trust

Real world example: www.datafix.com.

Problem We Solve

- ***Goal:*** maximize processing at *server*.
  - Minimize communication cost.
  - Key Holder (e.g., client) is resource-constrained.
- *Challenge:* query processing *without plaintext*
- Existing solutions: comparison & indexing.
  - E.g., *OPES* [AKSX04]. Directly compare/index ciphertext.
  - Can handle SQL query types *except SUM/AVG*.
  - Proposal of using homomorphism for SUM/AVG, but insecure [HIM04].

Missing: A comprehensive, secure solution for SUM/AVG

Outline

- System Model and Problem to Solve.
- Background.
- Algorithm 1: Basic Building Block.
- Algorithm 2: Handling Predicates and Compression.
- Algorithm 3: Randomized Pre-computation.
- Handling Floating Point Numbers.
- Experiments.
- Conclusions.

Homomorphic Encryption

- A well-known technique in *cryptography*.
- *Additive homomorphic:*

$$\boxed{\mathbf{enc(a + b) = enc(a) \times enc(b)}}$$

- *Generalized Paillier* cryptosystem.
  - Can adjust a parameter to make *ciphertext expansion factor* close to 1.

C-Store & Compression

- A *column-oriented* DBMS
  - *Read-optimized*, data warehousing applications

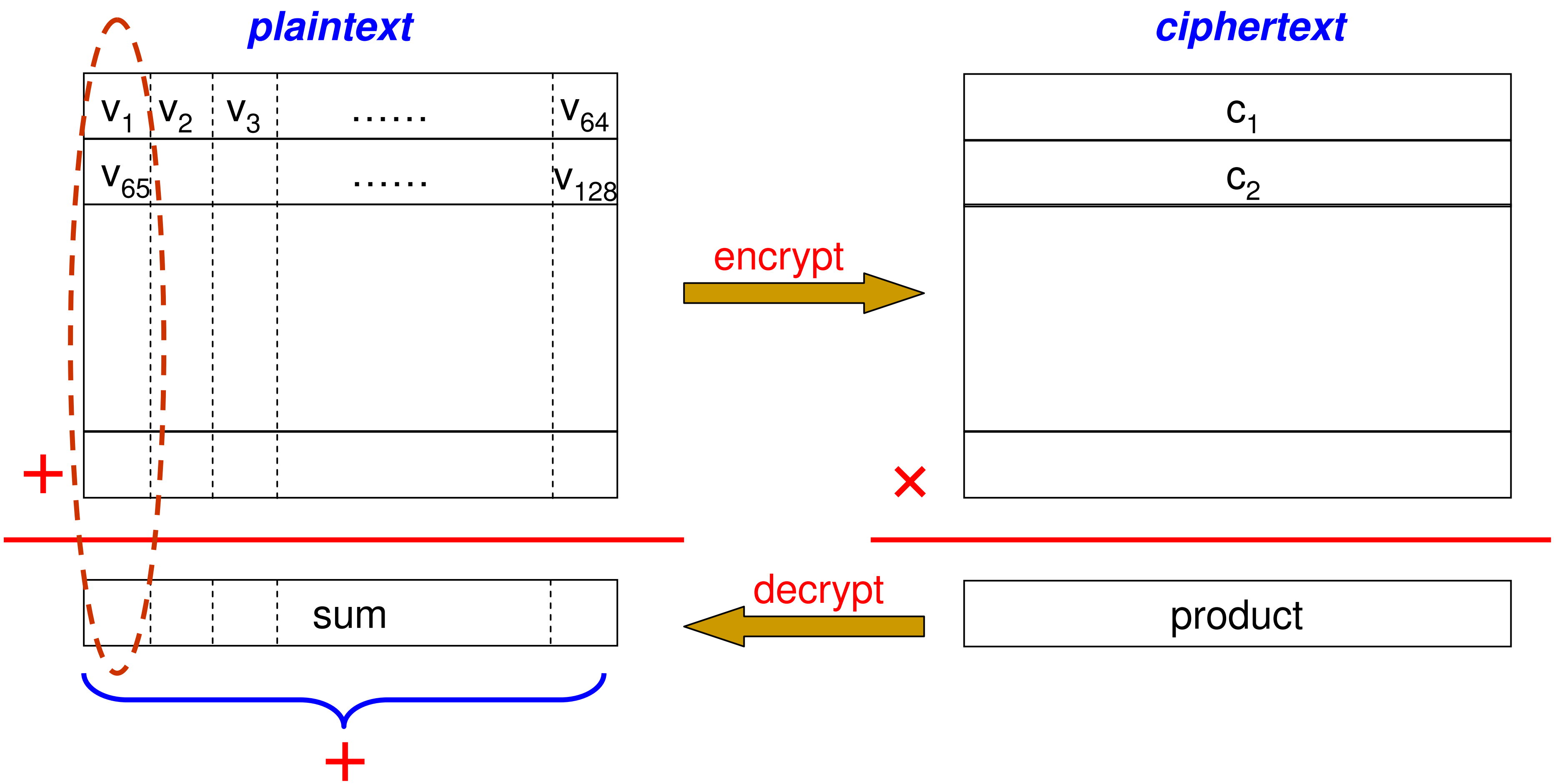
Name	SSN	Job	Salary
Alice	526-92	CEO	999,990
Bob	286-75	R&D	90,000
Cathy	756-98	Sales	99,000
Dan	892-16	Service	89,700
:	:	:	:
:	:	:	:

- Data can be *uncompressed* or *compressed*:
  - *Run length*, *bitmap*, and *delta* encoding

Outline

- System Model and Problem to Solve.
- Background.
- Algorithm 1: Basic Building Block.
- Algorithm 2: Handling Predicates and Compression.
- Algorithm 3: Randomized Pre-computation.
- Handling Floating Point Numbers.
- Experiments.
- Conclusions.

Basic Building Block

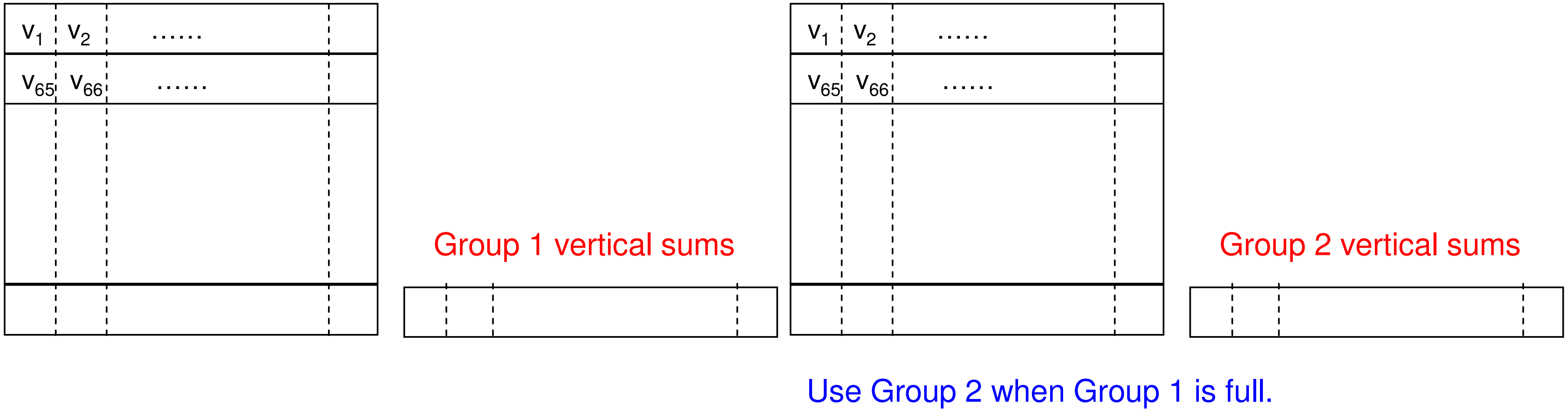


Algorithm 1

Assume, for now, no overflow.

Handling Overflows

- If overflow *within a vertical slice*, result wrong!
  - Arguably rare, but we need to handle it.
- Easy way: leave *extra space* preceding each plaintext value.
- Less easy way: *groups*; monitor sums.

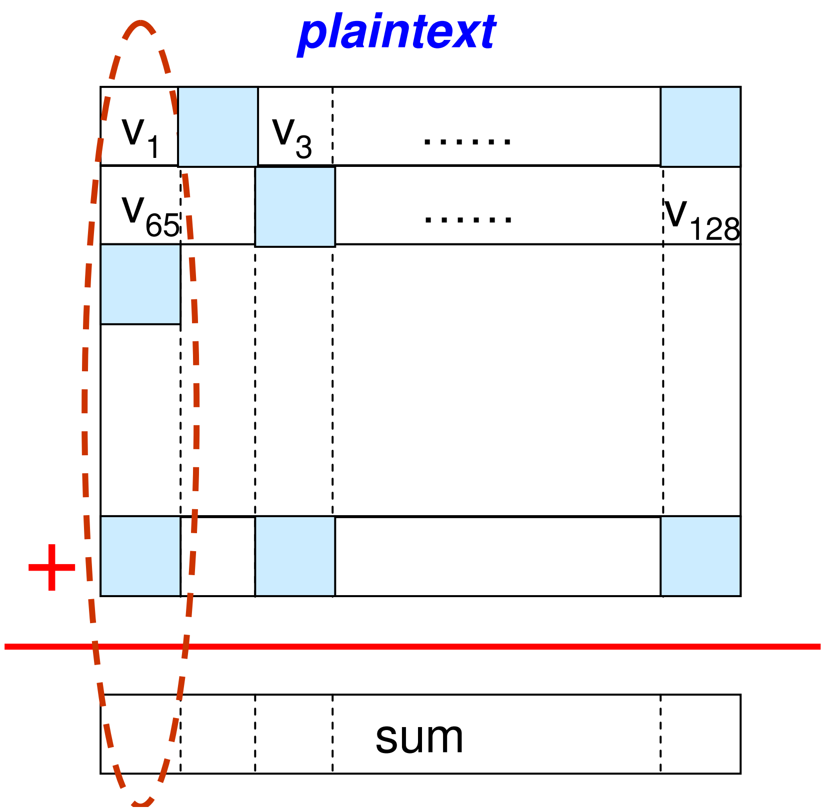


Outline

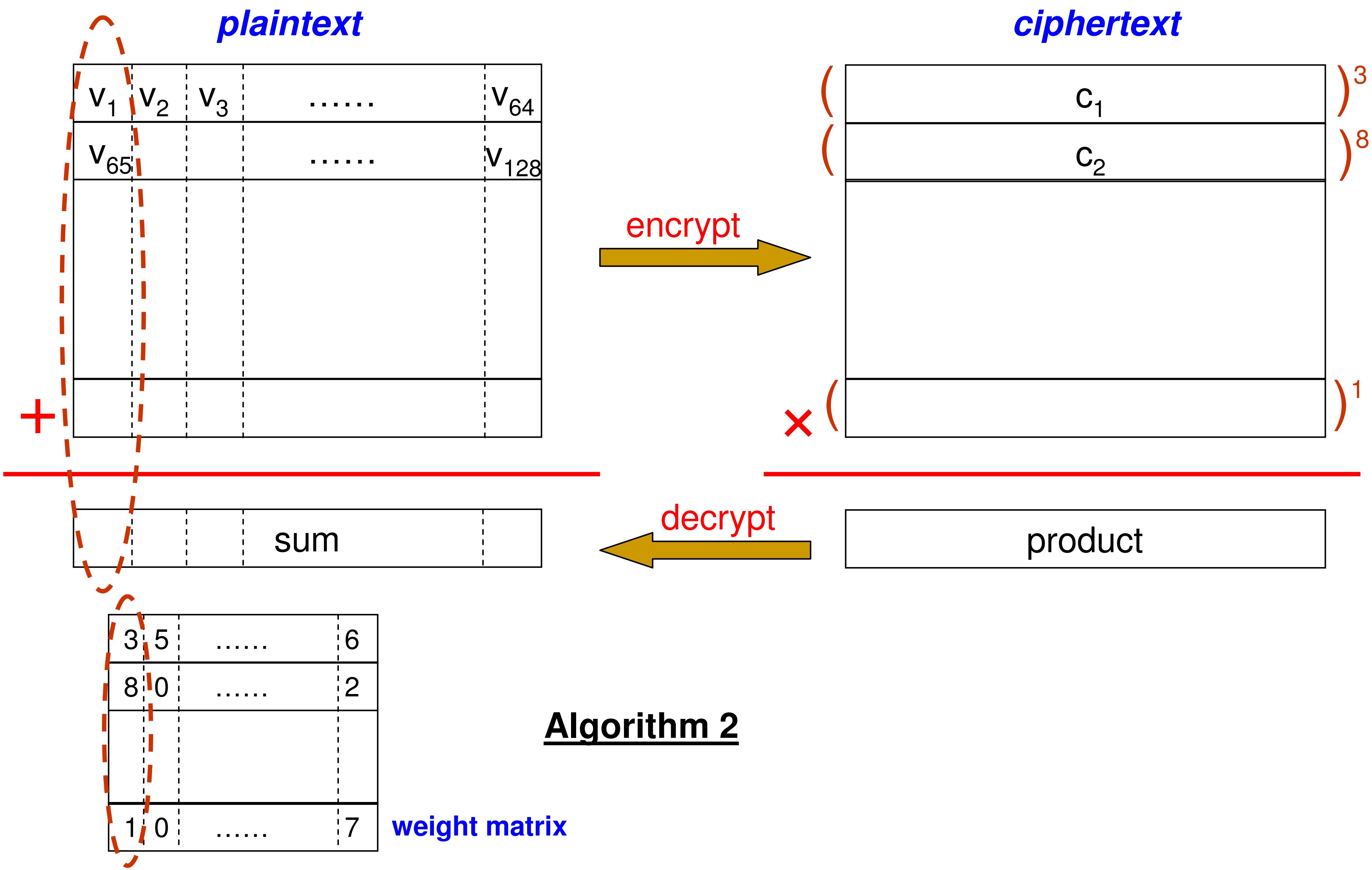
- System Model and Problem to Solve.
- Background.
- Algorithm 1: Basic Building Block.
- Algorithm 2: Handling Predicates and Compression.
- Algorithm 3: Randomized Pre-computation.
- Handling Floating Point Numbers.
- Experiments.
- Conclusions.

Why Is Alg. 1 Not Enough?

- Alg. 1 can only SUM/AVG *all column values*.
  - What about *SELECT AVG(salary) WHERE age > 25*?
  - SUM/AVG on C-Store *compressed data* directly?



An Extension of Algorithm 1



Algorithm 2

Handling Predicates

- Two categories of predicates:
  - Those that *do not* reference the *encrypted* column
  - Those that *do*.
- Q1: *SELECT **AVG(salary)** FROM employees WHERE **age > 35** AND **company = 'SUN'***
- Q2: *SELECT **AVG(salary)** FROM employees WHERE **salary > 60000** AND **company = 'MICROSOFT'***
- DBMS often compute a *bit-string* to represent the result of predicate (*1* if a record is qualified, *0* otherwise). The *bit-string* is a *binary weight matrix for Alg. 2*.
- For Q2, use *indexing on encrypted columns* (e.g., OPES). *salary* is encrypted *differently for SUM/AVG than in the index*.
- 2 predicates ➔ 2 bit-strings ➔ bitwise AND ➔ one bit-string

Update and Storage

- Insert new values incrementally, in *enc-block batches*.
  - OLAP, data warehousing (C-Store): *read-optimized, few or no updates, update in large batches.*

- A column can be *encrypted differently for SUM/AVG and for indexing*. Storage issue?
  - *Aggressive compression* in C-Store allows storing columns in different ways (e.g., sort orders).
  - Resort to a *sparse B+ tree* index: *sort before using homomorphic* encryption; then *sparse page-level index with OPES* (*first plaintext value* of each page enc'ed twice).
    - *SELECT AVG(salary) ….. WHERE* ***range-predicate-on-salary***
    - *Initial answer imprecise*; *post-process* 1st and last page of the range at Key Holder to make it precise.

Handling Compression

- **Run Length Encoding (RLE)**
  - *(value, # of repetitions)* pairs. Put all *value* parts in the *homomorphic enc blocks*. Put all *# of repetitions* parts in the *weight matrix*.
- **Bitmap encoding**
  - *(value, bitmap)* pairs. Put all *value* parts in the *homomorphic enc blocks*. <u>Count # of set-bits</u> in *bitmap* for the weights.
- **Delta encoding**

base	inc_1	inc_2

Decompressed values: base $base+inc_1$ $base+ inc_1 +inc_2$

$base \times 3$ $inc_1 \times 2$ $inc_2 \times 1$

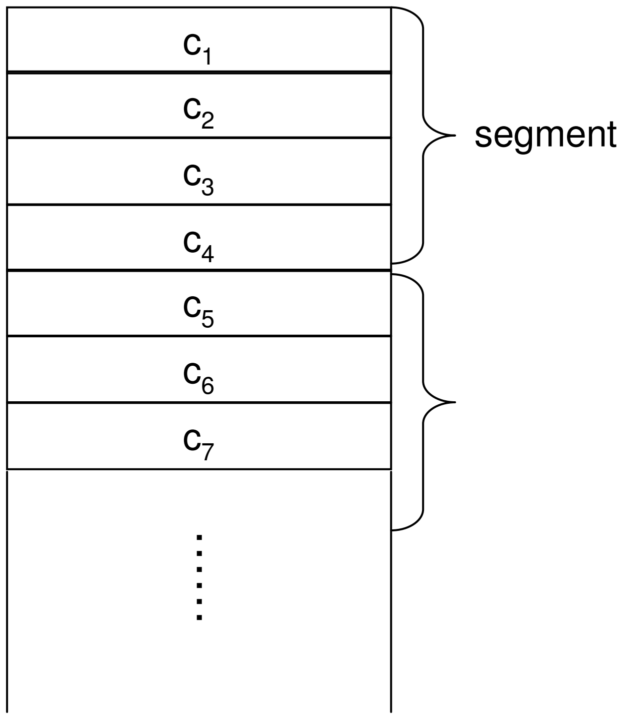
Put in homomorphic enc. block

Weights for Alg. 2

Outline

- System Model and Problem to Solve.
- Background.
- Algorithm 1: Basic Building Block.
- Algorithm 2: Handling Predicates and Compression.
- Algorithm 3: Randomized Pre-computation.
- Handling Floating Point Numbers.
- Experiments.
- Conclusions.

A Randomized Pre-computation



Pre-compute modular product of random subsets:

c_2c_4, $c_1c_2c_3$, c_1c_4, c_2c_3.

Use same amount of space as ciphertext.

Suppose weight matrix for some query's predicates:

$$\begin{matrix} 1 & 0 & 1 & 1 & \ldots\ldots \\ 0 & 1 & 0 & 1 & \ldots\ldots \\ 1 & 1 & 1 & 0 & \ldots\ldots \\ 1 & 0 & 1 & 0 & \ldots\ldots \end{matrix}$$

1st column: $c_1c_3c_4 = (c_1c_4) \times c_3$

2nd column: c_2c_3 available.

3rd column: $c_1c_3c_4$ available from 1st column.

4th column: $c_1c_2 = (c_1c_2c_3) / c_3$.

Algorithm 3

Determine When to Use Algorithm 3

- If the combined *selectivity* of all predicates is *p*, the *fraction of multiplications* of *Algorithm 2* is *1/p*; the fraction of *Algorithm 3* is $\leq \frac{E(M)}{k \cdot s} + \frac{1}{s}$.

  E(M): expectation of # of multiplications per segment.

  k: # of values per encryption block.

  s: segment size.

  - If *k = 64, s = 7*, on average, *Alg. 3* performs better *when p is greater than 0.27*.

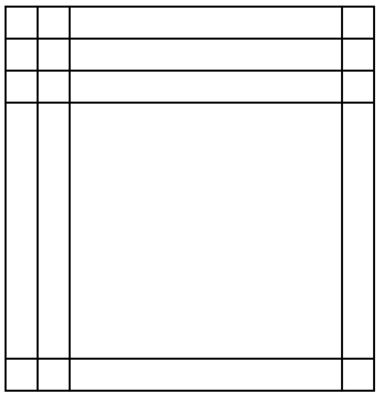
- During execution, *from the weight matrix, we know p*, and *decide whether to kick off Algorithm 3* or just use Algorithm 2.

Outline

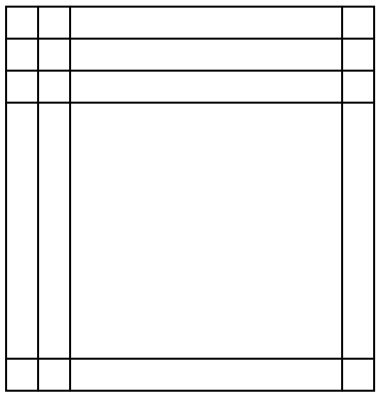
- System Model and Problem to Solve.
- Background.
- Algorithm 1: Basic Building Block.
- Algorithm 2: Handling Predicates and Compression.
- Algorithm 3: Randomized Pre-computation.
- Handling Floating Point Numbers.
- Experiments.
- Conclusions.

On Floating-Point Numbers (IEEE 754 Standard Single Precision FP)

- Observation: If we add two numbers that *differ at least 24 in exponents*, the result is simply the bigger number.
- Basic idea: Have multiple ciphertext *groups*, each containing values *within a "24" exponent range*. Pick one group to use.



G_0: for SUM of a list of numbers with max exp. in *[248, 255]*. G_0 contains all column values with exp. in *[248 – 23, 255]*, normalized to 248.



G_1: for SUM of a list of numbers with max exp. in *[240, 247]*. G_1 contains all column values with exp. in *[240 – 23, 247]*, normalized to 240.

......

32 groups, each covering a range of 8 for max exp.

Which Group to Use for a Query?

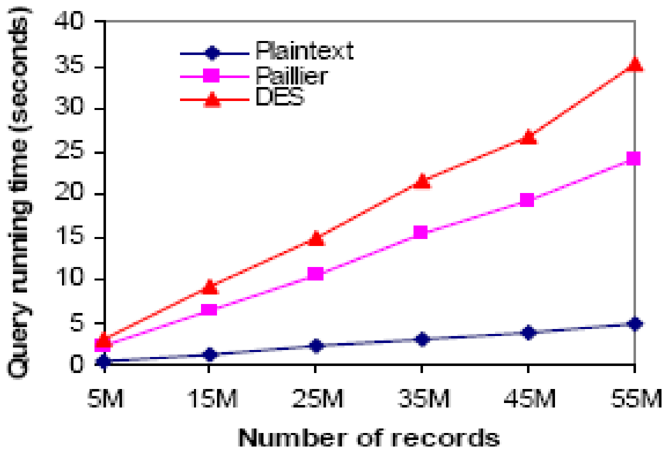
- Use *bitmaps*, representing a set of records.
- Each group: a bitmap M_i showing which records are in its "*max exp. range*".
- Evaluating predicates of a query gives a bitmap P. Find the *1st group* whose M_i *intersects* P.
- Each group: another bitmap T_i showing which records are in its whole *"24" exponent range*.
  - *ANDing* P *and* T_i gives the *weight matrix*.

Outline

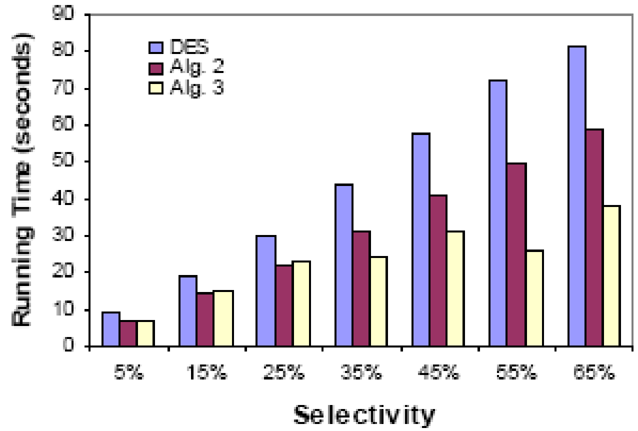
- System Model and Problem to Solve.
- Background.
- Algorithm 1: Basic Building Block.
- Algorithm 2: Handling Predicates and Compression.
- Algorithm 3: Randomized Pre-computation.
- Handling Floating Point Numbers.
- Experiments.
- Conclusions.

Experiments

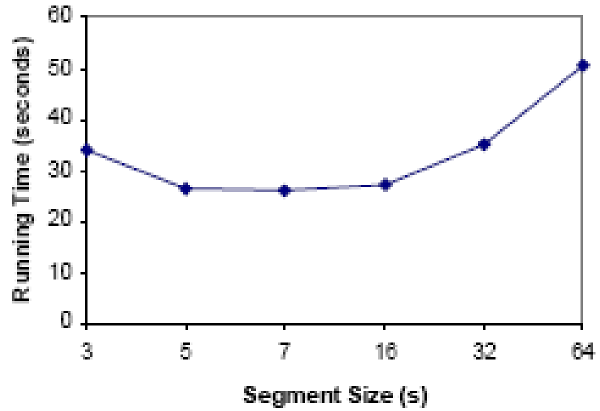
Goal: To verify that the performance is acceptable, as the only solution.



Performance of Alg. 2 and comparisons. (SELECT AVG with a range predicate, 25% selectivity).



50M records, different selectivities.



Alg. 3 with different segment sizes, with selectivity fixed at 50%.

Conclusions

- Proposed techniques to *answer SUM and AVG* queries in a secure model without decryption key.
- Handle arbitrary *predicates and compression* schemes of C-Store.
- Combined with other schemes that handle *comparison and indexing*, we approach a nearly complete solution.
- Proposed a *randomized pre-computing* technique to further improve performance.
- Verified that *performance is competitive*.

Thank you !!

- Questions ?