Supporting Joins and Numerical Computations over Encrypted Databases

Department of Information Engineering and
Computer Science
University of Trento

Alex Pellegrini

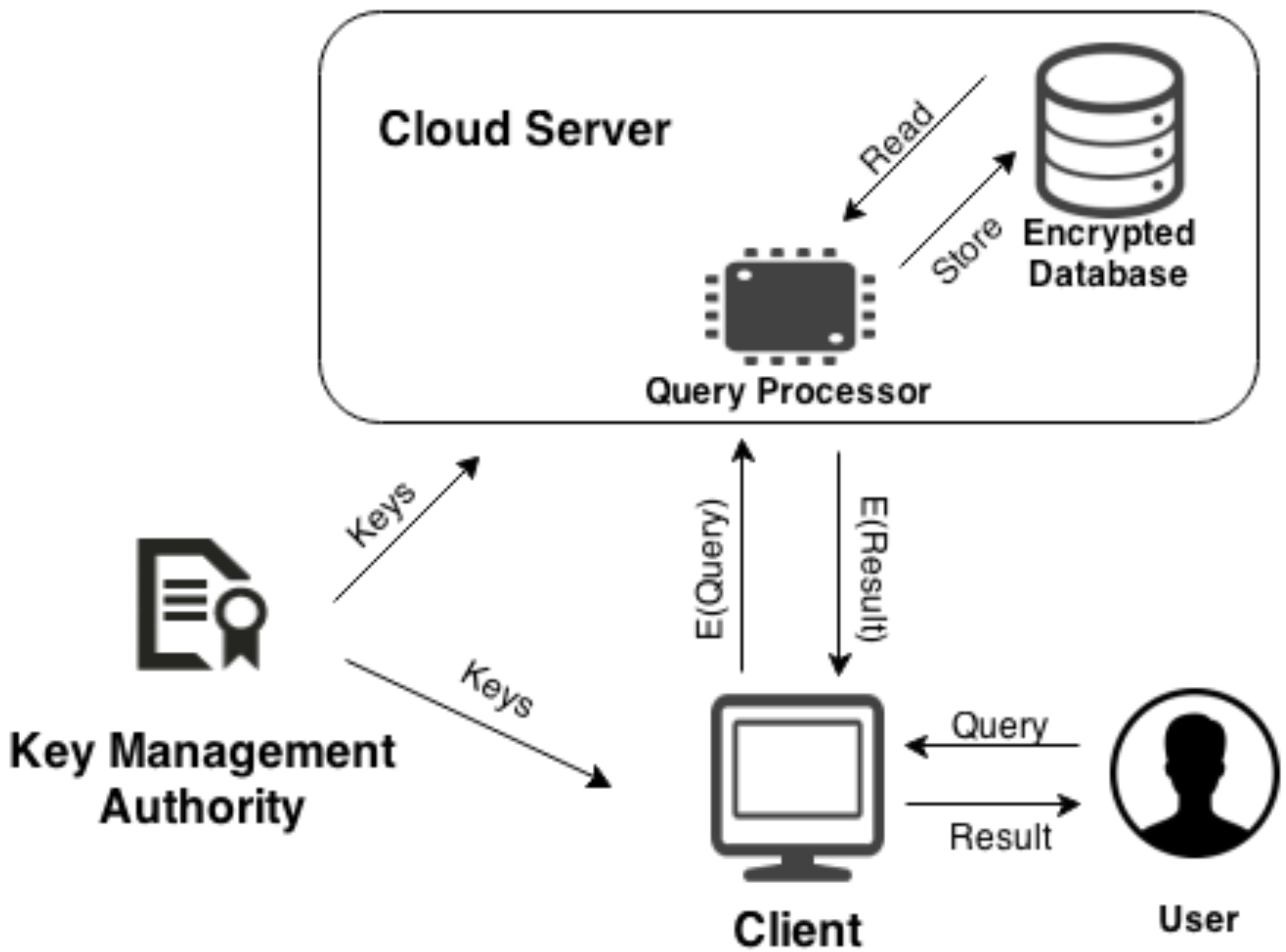
Supervisors:
Associate Prof. Dr. Bruno Crispo
Dr. Muhammad Rizwan Asghar

September 2014

Background

- Everyday growing data
- Data outsourcing
  - Unauthorized accesses and attacks
  - Security and confidentiality challenges

CloudDB



Cloud Server
Read
Store
Encrypted Database
Query Processor
Keys
Keys
E(Query)
E(Result)
Key Management Authority
Query
Result
Client
User

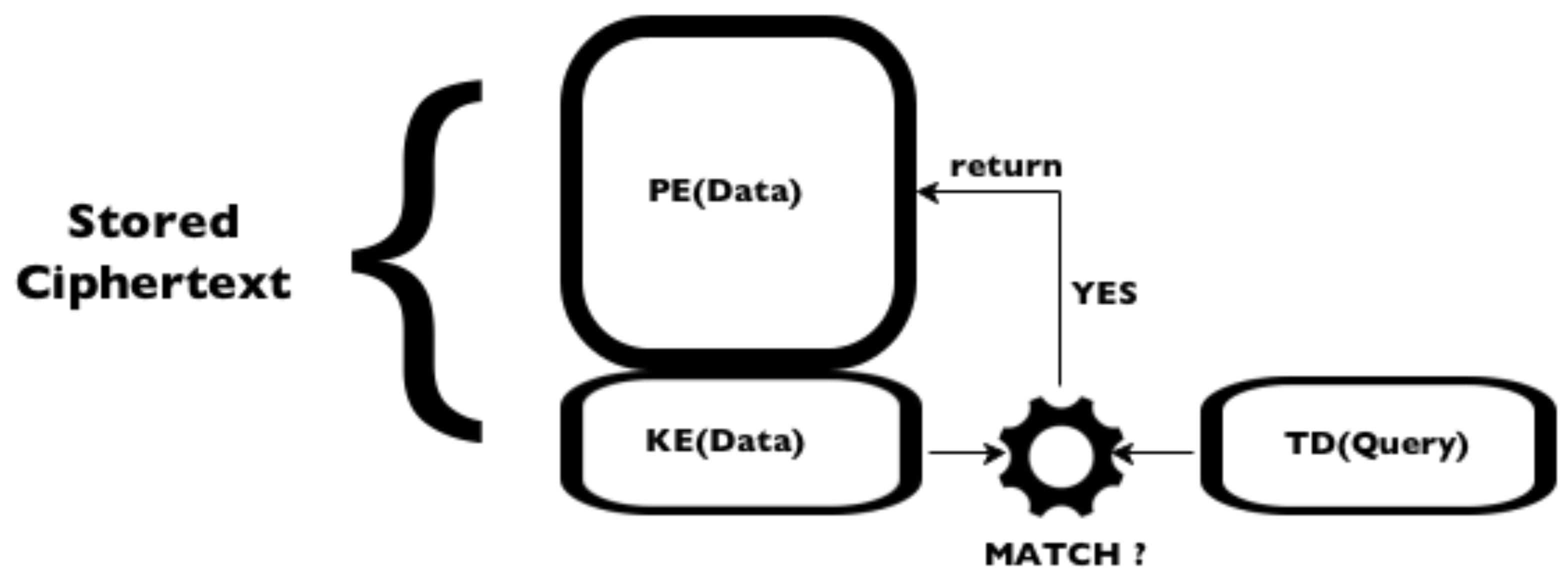
Encryption Schemes

- Data Retrieval
    - Proxy Encryption (*PE*)
- Data Search
    - Keyword Encryption (*KE*) :: Data searchable
    - Trapdoor Encryption (*TD*) :: Query (read) Encryption

Encrypted Match

function PE[] SEARCH(*TD(Q), Data*)
  PE[] *matching* ← ***new PE***[*Data.size*]*;*
  for all *D* in *Data* **do**:
    if MATCH(*TD(Q), KE(D))*:
      matching.append(PE(D));
    end if
  end for
  return *matching;*
end function

Match(KE(D), TD(Q)) → {0, 1}



Goals

- Support computations between numerical ciphertexts
- Evaluate range SQL encrypted queries on numerical ciphertexts
- Combine encrypted records to join tables

Numerical Data

- Introduction of `Integer`(s) data type.
- Probabilistic Homomorphic Encryption (*HE*)
  - Adapted Paillier Cryptosystem
- *PE* is replaced with *HE* for numerical data

HE : Encryption

- x : secret key from Z^*_q
  n : product of two large primes
  g : an nth-residue of $Z^*_{n^2}$
  D : numerical value
  r_D : random number
- Compute :
  $C_{D_1} = \{ c'_1 = g^{r_{D_1}}, c''_1 = g^{x r_{D_1}}(1 + D_1 n) \}$

HE : Sum

- $C_{D_1} = \{ c'_1 = g^{r_{D_1}}, c''_1 = g^{x r_{D_1}}(1 + D_1 n) \}$
  $C_{D_2} = \{ c'_2 = g^{r_{D_2}}, c''_2 = g^{x r_{D_2}}(1 + D_2 n) \}$
- Sum (element-wise product):
  $C_D = C_{D_1} C_{D_2} = \{ c' = c'_1 c'_2, c'' = c''_1 c''_2 \} =$

$$\{$$
$$c' = g^{r_{D_1} + r_{D_2}},$$
$$c'' = g^{x(r_{D_1} + r_{D_2})}(1 + D_1 n + D_2 n + D_1 D_2 n^2)$$
$$\}$$

HE : Decryption

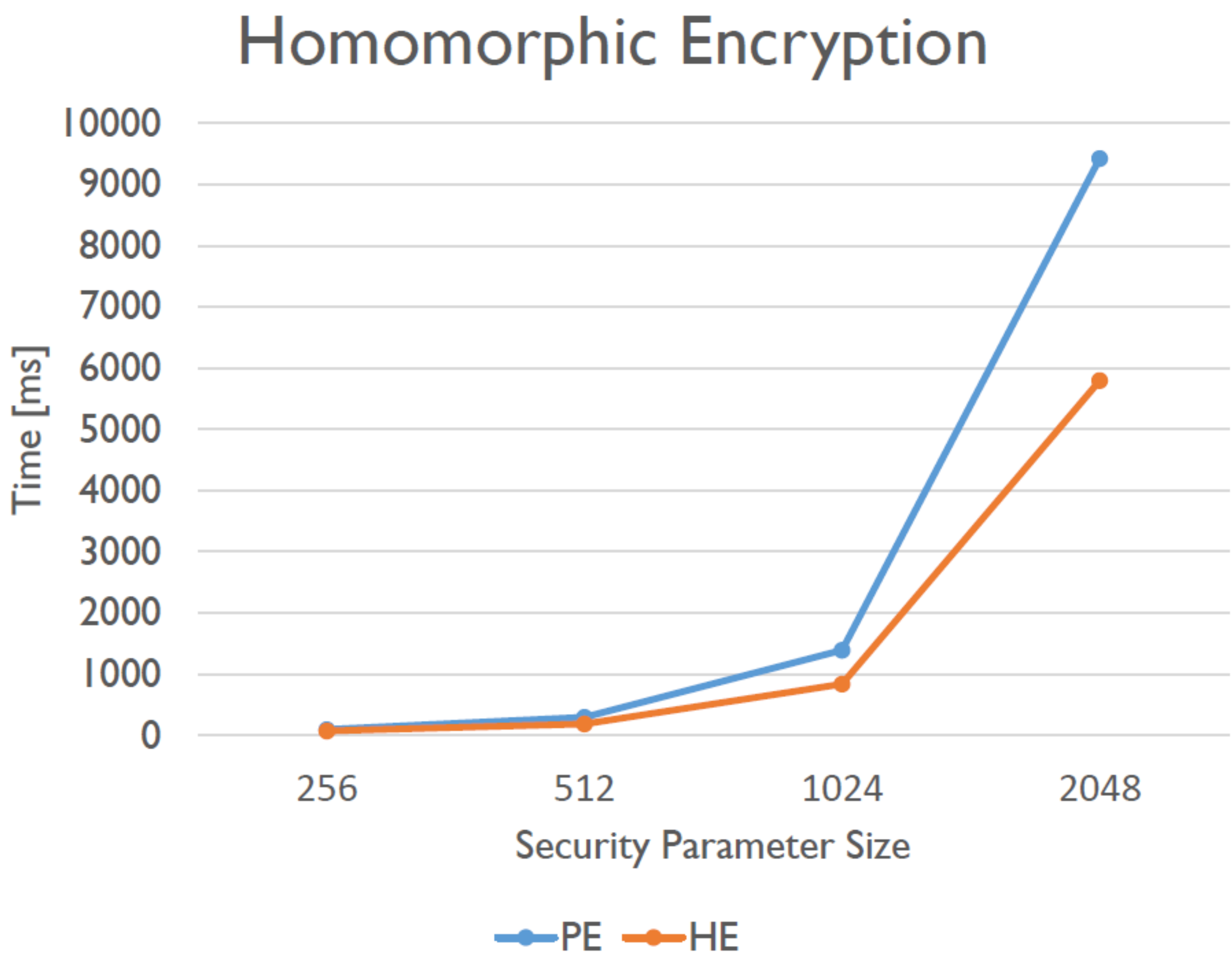
- Decrypt $C_D = \{ c', c'' \}$:

  $\lambda = c''(c')^{-x} = (1 + D_1 n + D_2 n + D_1 D_2 n^2)$

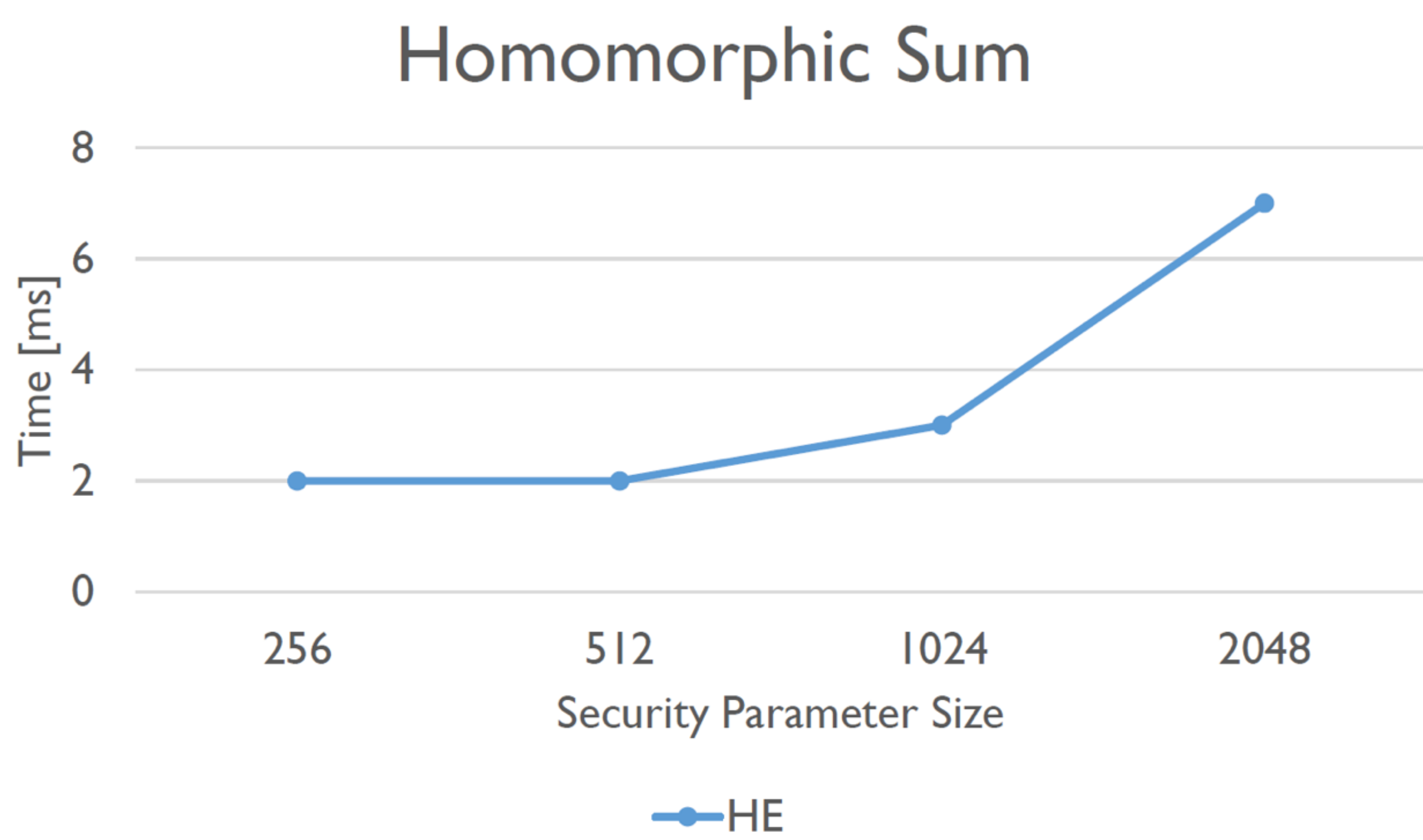
- Compute D :

  $(\lambda - 1) / n \bmod n = D_1 + D_2 = D$

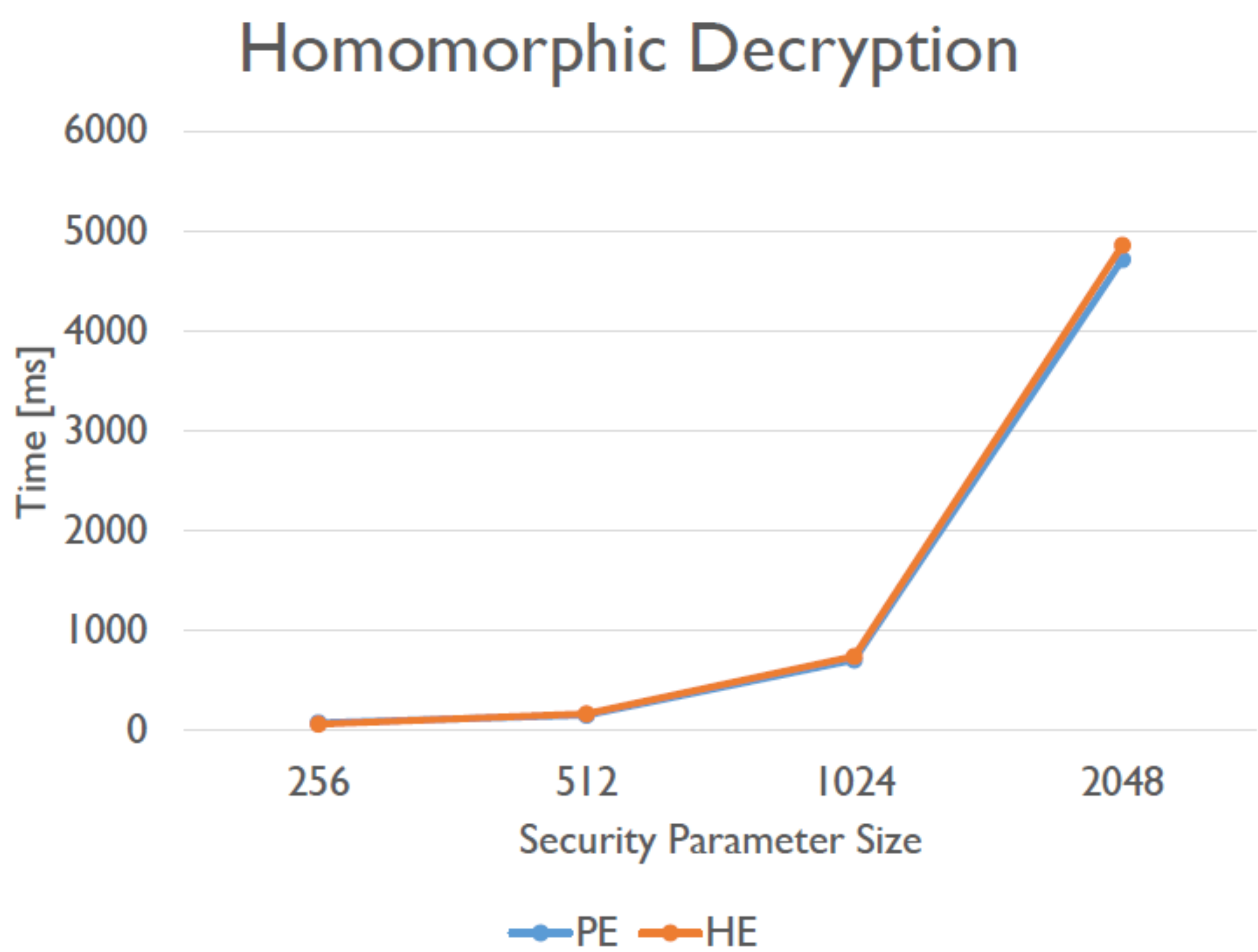
HE : Encryption Performances



HE Sum Performances



HE : Decryption Performances



Range Queries Evaluation

- Requirements :
  - Numerical columns' bit length known a priori
  - A supplementary table related to numerical columns
  - The ***Bag of Bits*** approach

Bag of Bits

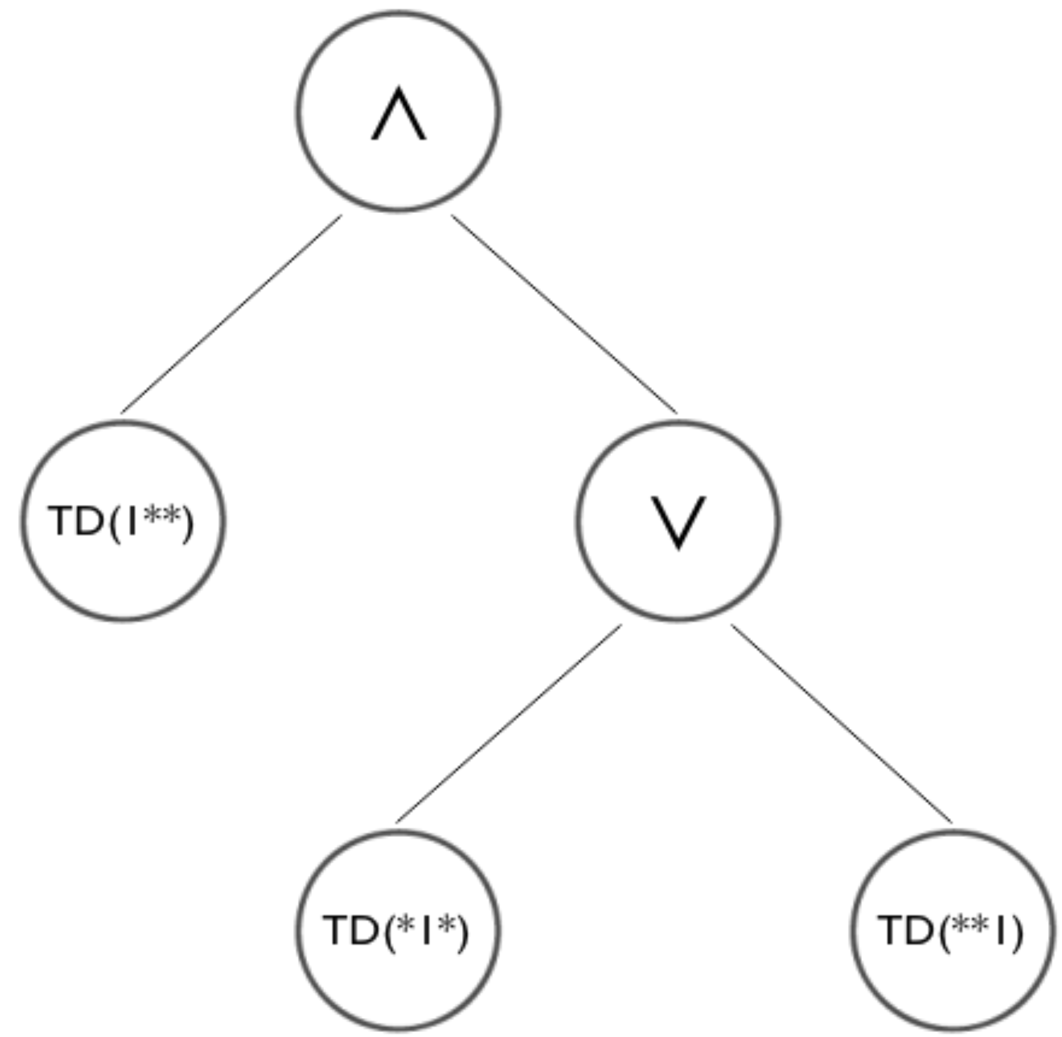
- A set (for each value) made up by *KE* ciphertexts of strings composed by every bit of the value and filled up with *
- k = **3**,
  n = **5**, n_2 = **101**
- Bag of Bits for n is then made up by :

  { KE(**1****), KE(***0***), KE(****1**) }

Condition Tree

- `n > 4;`
- `5 = 101,`
  `6 = 110,`
  `7 = 111`

  1** AND (*1* OR **1)



Join

- Data Stored as *PE/HE* and *KE* (probabilistic schemes)
- No way to compare stored data.

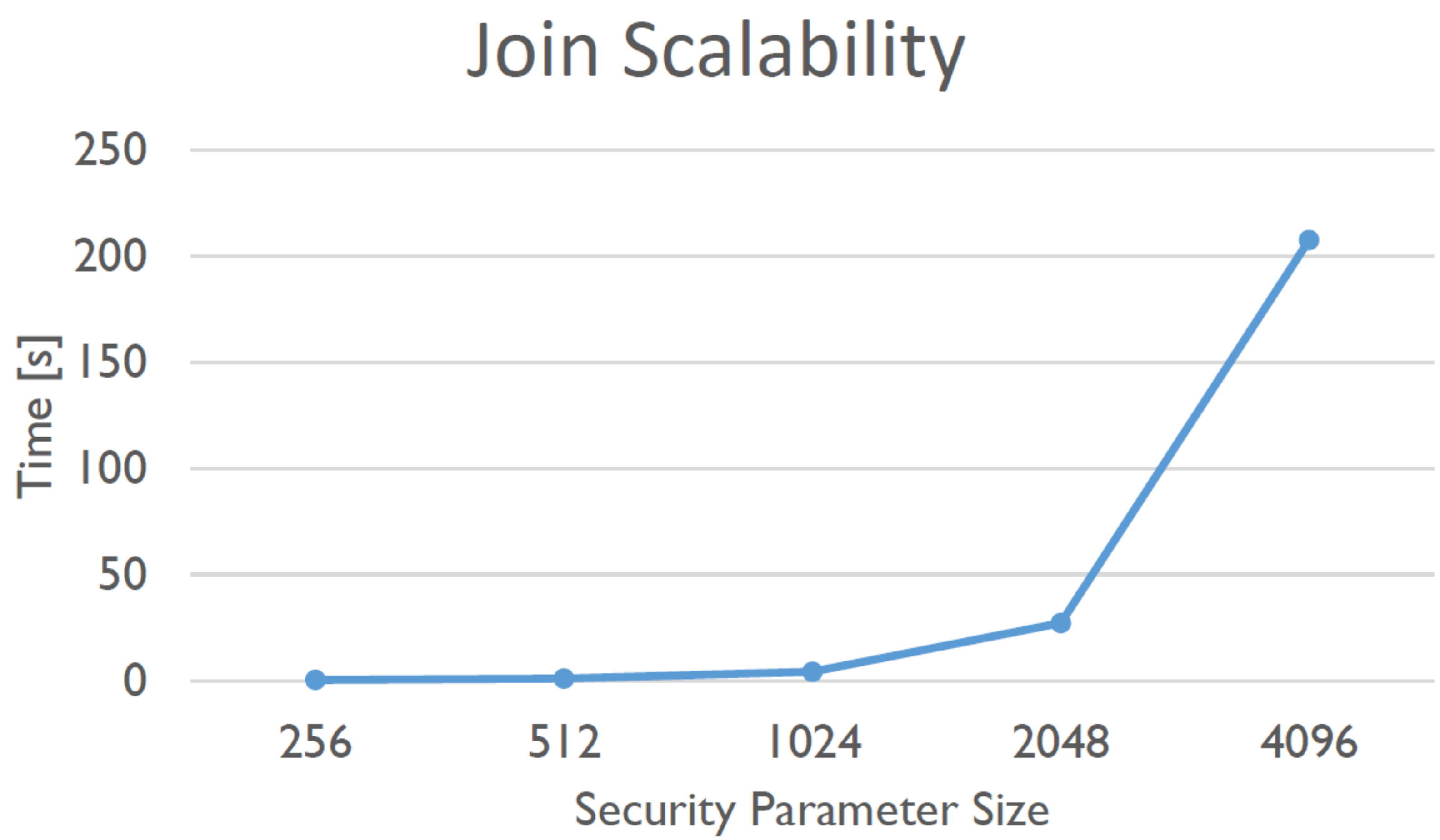
Cross Join

- No constraints on field values
- Combine records of a table with records of other tables (Cartesian Product)
- The DBMS can do the work.

ON Policy

- Joinable columns known a priori
- Store values in joinable columns as *PE/HE* and *TD* ciphertexts
  - No need of *Match* function evaluation
  - String equality comparison (very fast)

Join Performances



Join between two tables of 3 and 2 columns, 100 rows each and 100 matches. Result of 100 rows and 5 columns.

Conclusions

- Relatively fast Paillier Cryptosystem adaption
- Bag of Bits approach for numerical range queries
- Easy and fast Join solution

That’s it !

Thank you for paying attention.

Alex