Toward Privacy-Assured Cloud Data Service

Wenjing Lou

Complex Networks and Security Research Lab (CNSR)

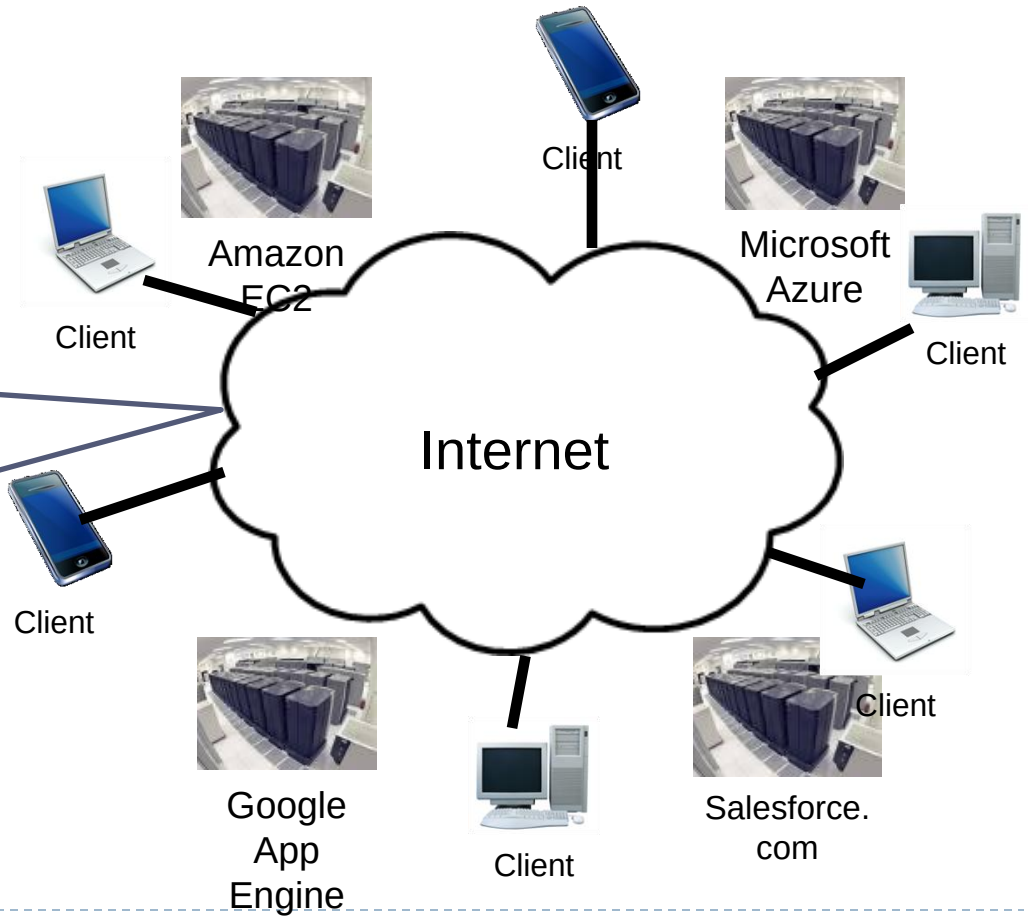
Virginia Polytechnic Institute and State University

What is Cloud Computing?

- Cloud Computing is the long dreamed vision of computing as a utility.

Cloud: data center hardware and software

Utility computing: applications delivered as services over the Internet



XaaS:
Software as a Service
Infrastructure as a Service
Platform as a Service

Research Challenges on Security

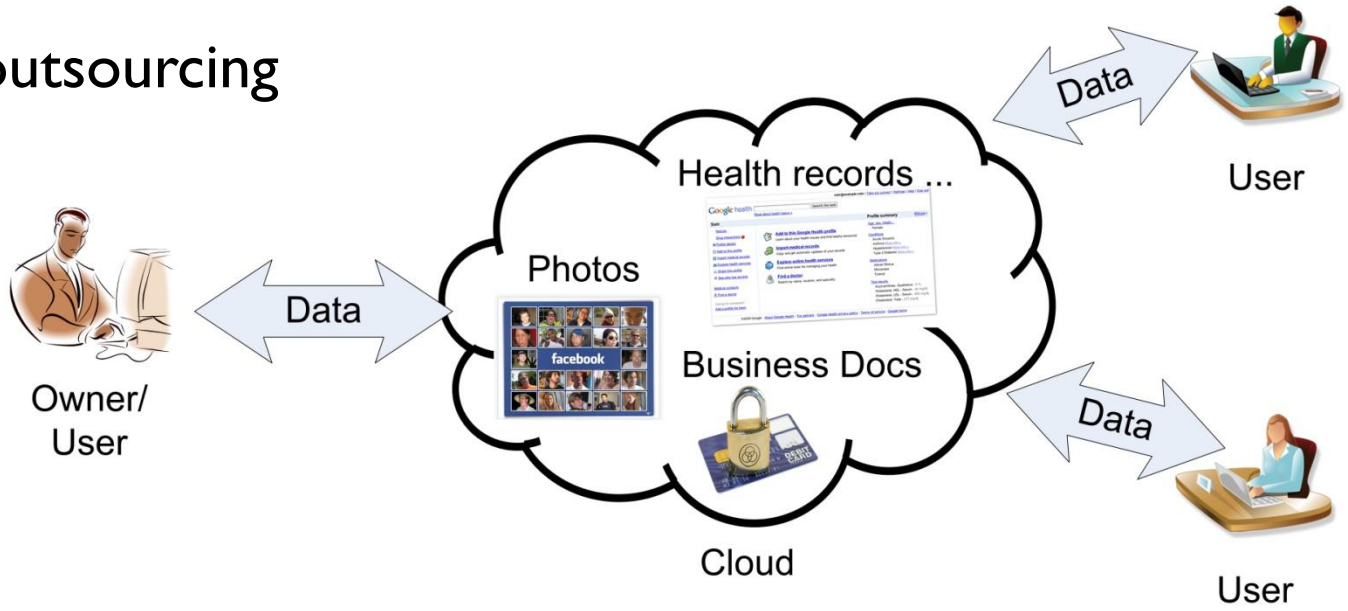
Moving into Cloud = Lost of Control

- Secure resource virtualization
- Privacy-assured data services
- Secure computation outsourcing
- Business and security risk models
- Service and data auditing

- Some key issues
  - trust, multi-tenancy, encryption, compliance…

Data Security Model

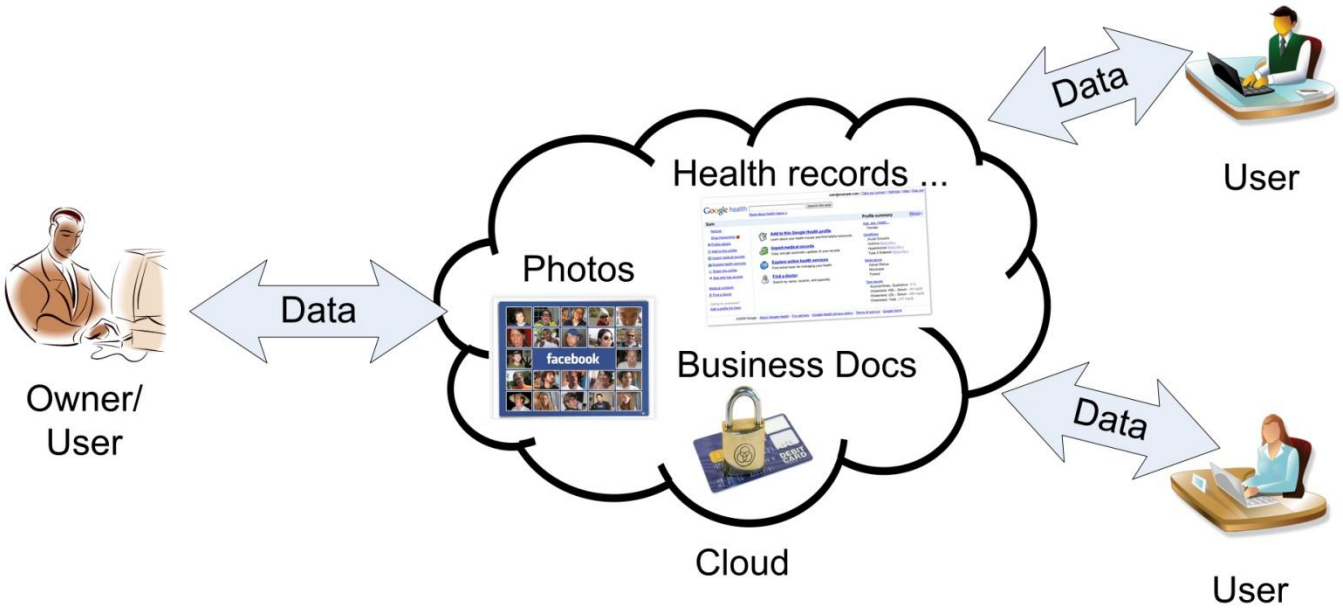
Data outsourcing



- Confidentiality & access control: read/white/execute, protection of data from unauthorized disclosure and use
- Integrity & availability: assurance of data stored on the server are genuine, correct, and complete (i.e., no fabrication, no corruption, no lost)
- Secure sharing and utilization …

What is new?

Shall We Trust the Cloud Server?



Trusted server (on company's premise)

vs.

Untrusted servers (third-party cloud service providers)

Untrusted server ⇨ new security research paradigm

New Security Problems

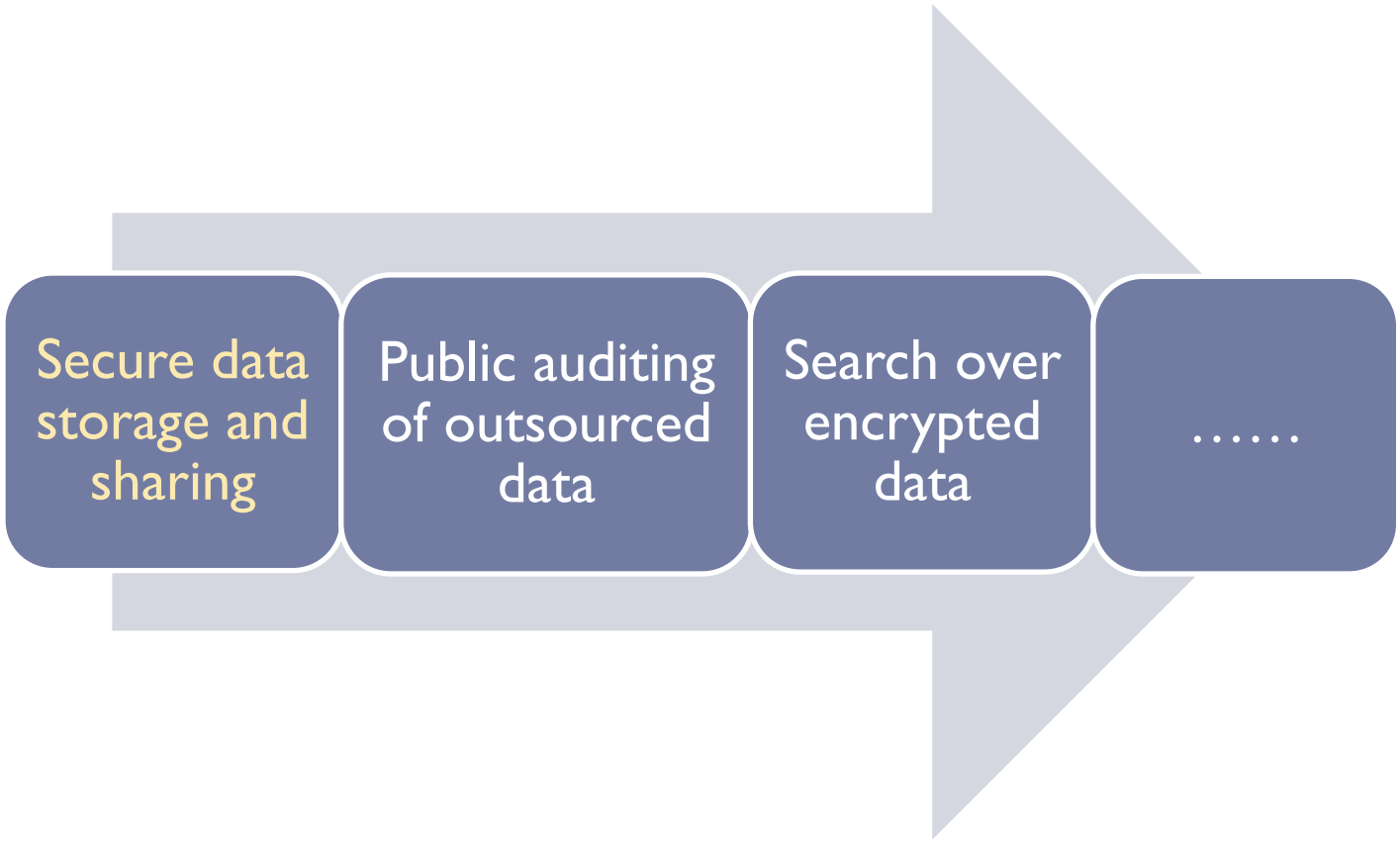
- Confidentiality
  - Access control? Traditional solutions rely on server enforce the policies
- Integrity
  - You do not have a local copy of the data
- Security of outsourced computation
- Auditing
  - Customer/third-party auditing of the utility metering

- Virtualization security
  - Shared resource environment => application confinement, side channel, covert channel
- Monitoring perspective and attack detection
  - Traffic concentration point
- Trustworthy architecture for secure cloud servers and end-user devices

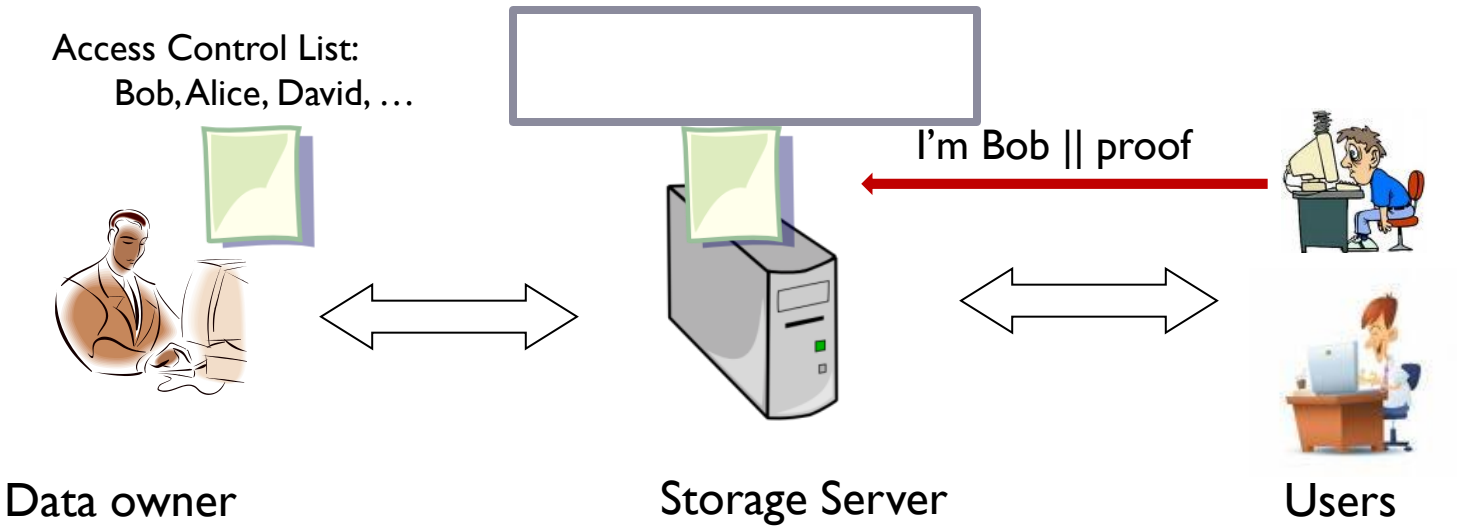
Privacy

- Privacy is a bigger problem in cloud computing
- Huge amount of data are in the cloud and will be there indefinitely
- Encryption before outsourcing => utilization problem
  - Effective data sharing ? --- encryption schemes and key management
  - Search over encrypted data
  - General processing over encrypted data --- homomorphic encryption system
  - Efficiency of the solution
- Privacy protection by non-crypto means
  - Distributing data onto multiple clouds

Research on Privacy-Assured Data Service

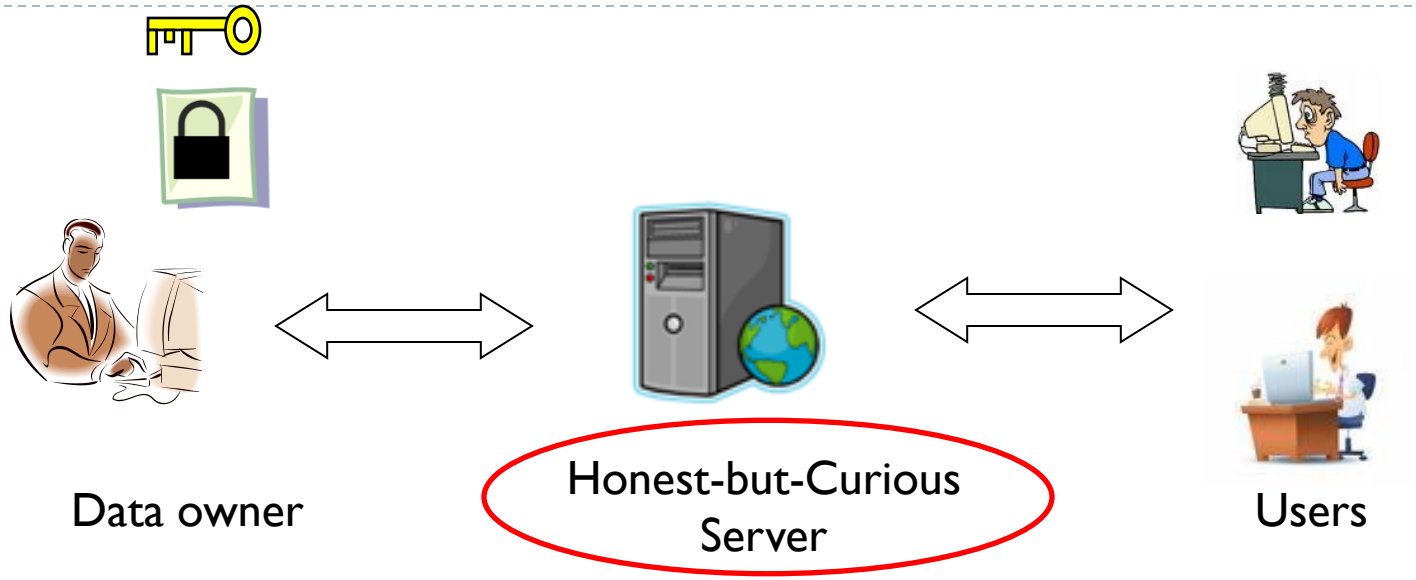


Traditional Server Mediated Access Control



- Data owner would store sensitive data on the server
- **Fine-grained access control** means different users would have different access privileges (read/write) over the records/files that they are allowed to access
- Trusted server to enforce the ACL policies

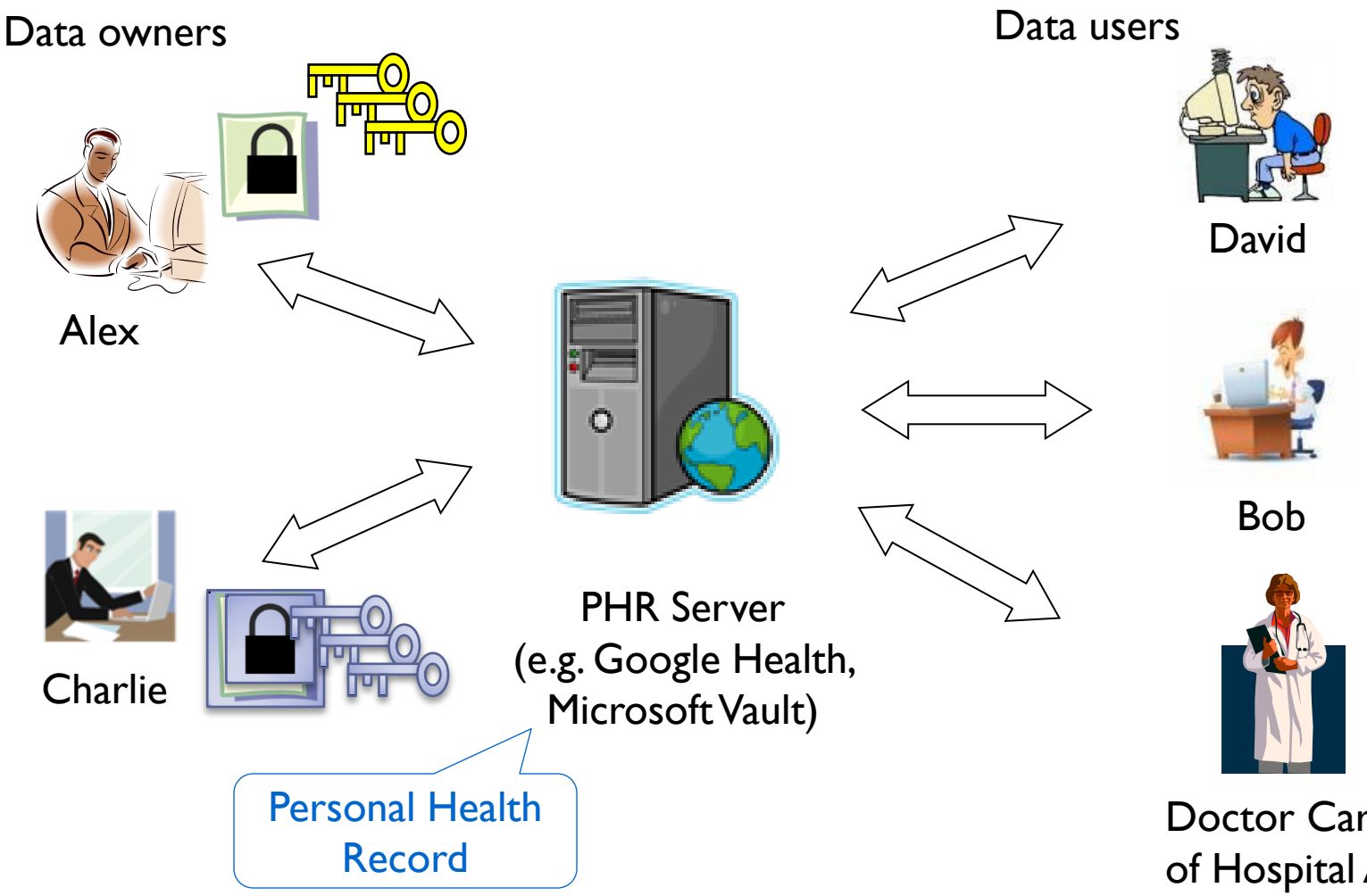
Encryption-Based Access Control



Confidentiality against server

Encryption before outsourcing

System Model: multi-owner multi-user



(Data users may collude with each other)

Challenges for Encryption-based Access Control

- Fine-grained access control with system scalability
  - Single key for all files: no fine grained-ness
  - New key for each file: system scalability
- User dynamics
  - User access privilege grant and revocation
- Effective data utilization
  - Search/record matching

Choice of Encryption Scheme

- Symmetric key encryption [Kallahalla03], [Damiani05], [Vimercati07], [Wang09]
  - Pros: Efficient for data encryption, can realize access control lists
  - Cons: Key management complexity, for large-scale systems, need an online trusted party to distribute keys, not collusion-resistant, users' read/write rights are not separable, user list must be known in advance

- Public key encryption or identity-based encryption [Goh03], [Ateniese05], [Hwang 07] , [Benaloh09]
  - Pros: Good for user authentication, scalable in user key management – one private key per user
  - Cons: Encryption complexity linear with number of users (one encryption for each user), need to know authorized users before encryption

Attribute-based Encryption (ABE)

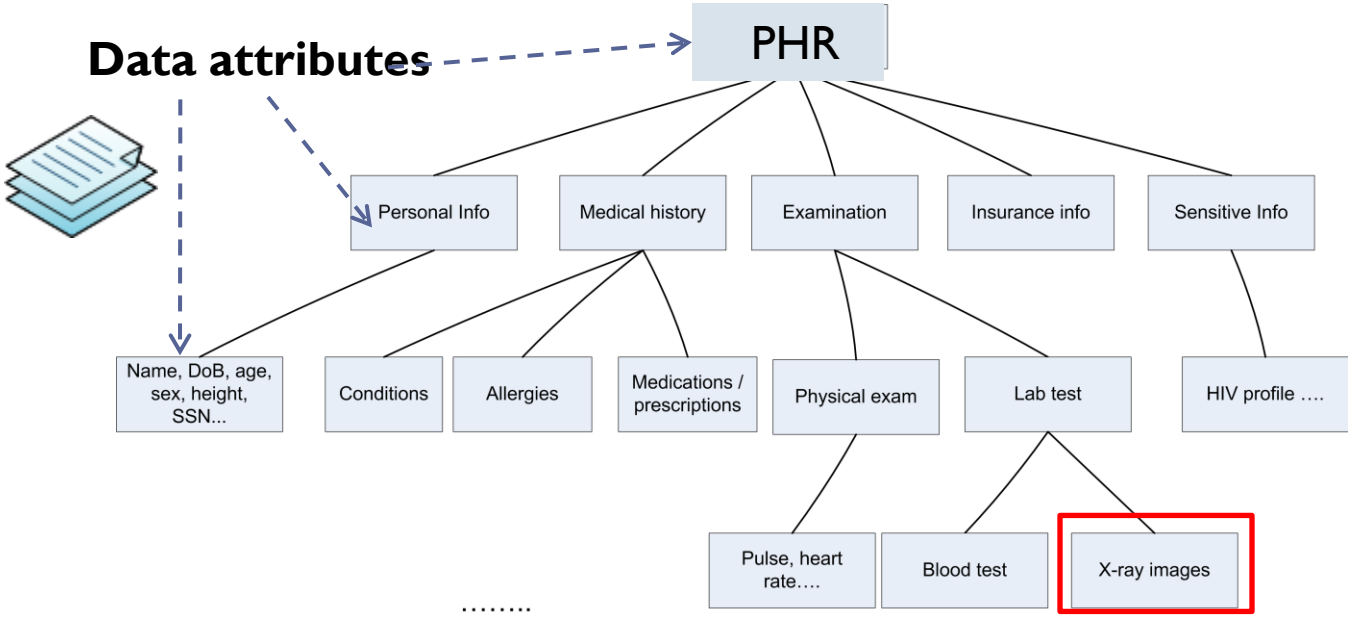
- A generalization of identity-based encryption
  - IBE: one public key, one master private key
  - ABE: multiple public/private keys (i.e., attributes), allows complex rules specifying which private keys can decrypt which ciphertexts

- Key-Policy ABE (KP-ABE) [Goyal06]
  - Ciphertexts are associated with "attributes"
  - Decryption policies (i.e., access structures) are embedded in private keys

- Ciphertext-Policy ABE (CP-ABE) [Bethencourt07]
  - Private keys are associated with "attributes"
  - Decryption policies (i.e., access structures) are embedded in ciphertexts

Types of Attributes in PHR systems

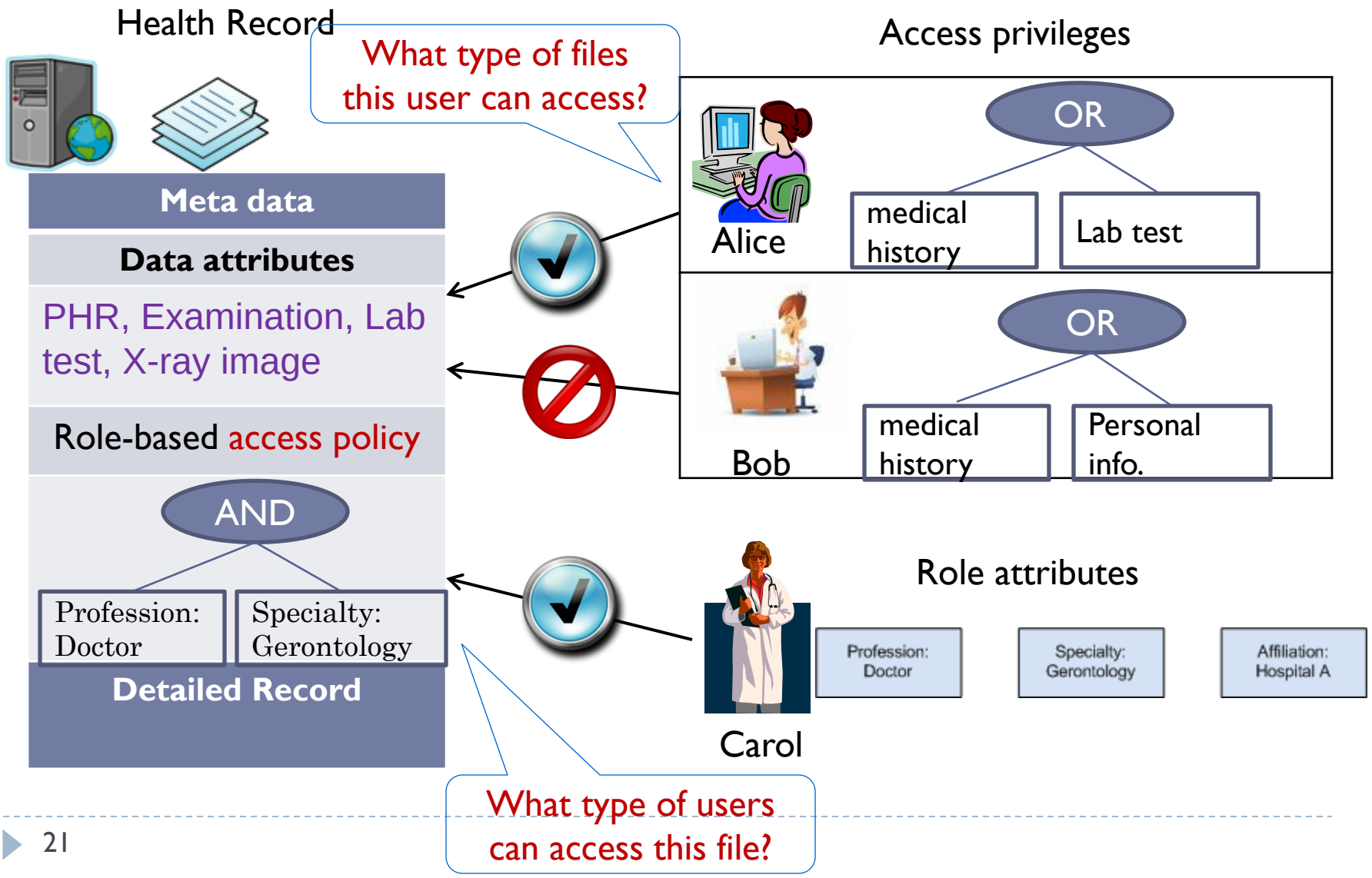
Data attributes



A= {PHR, Examination, Lab test, X-ray images}

Role attributes

Data Access Privileges /Access Policy



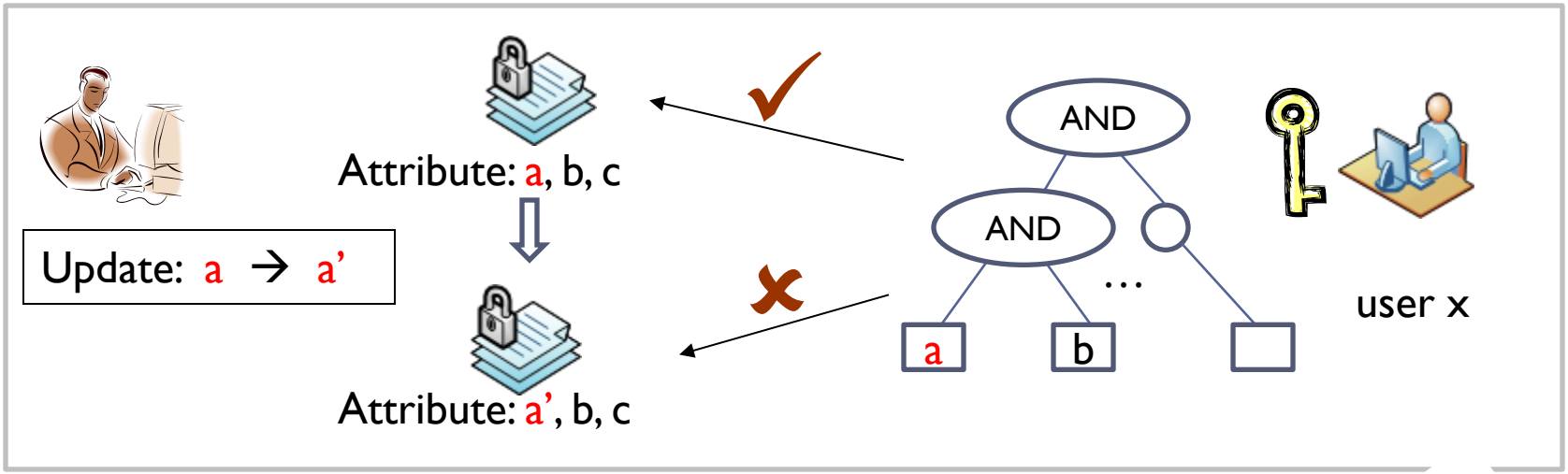
ABE-based Data Access Control

- Fine-grained data access control: access structure can be any Boolean formula
- Collusion-resistance: a property of ABE
- Scalability: complexity of encryption/decryption linear to # of attributes/access structure size

- Main techniques used:
  - ABE (KP-ABE, CP-ABE, MA-ABE)

Efficient User Revocation for ABE

- **Basic Idea:** update a minimal set of attributes to disable the user's access structure.



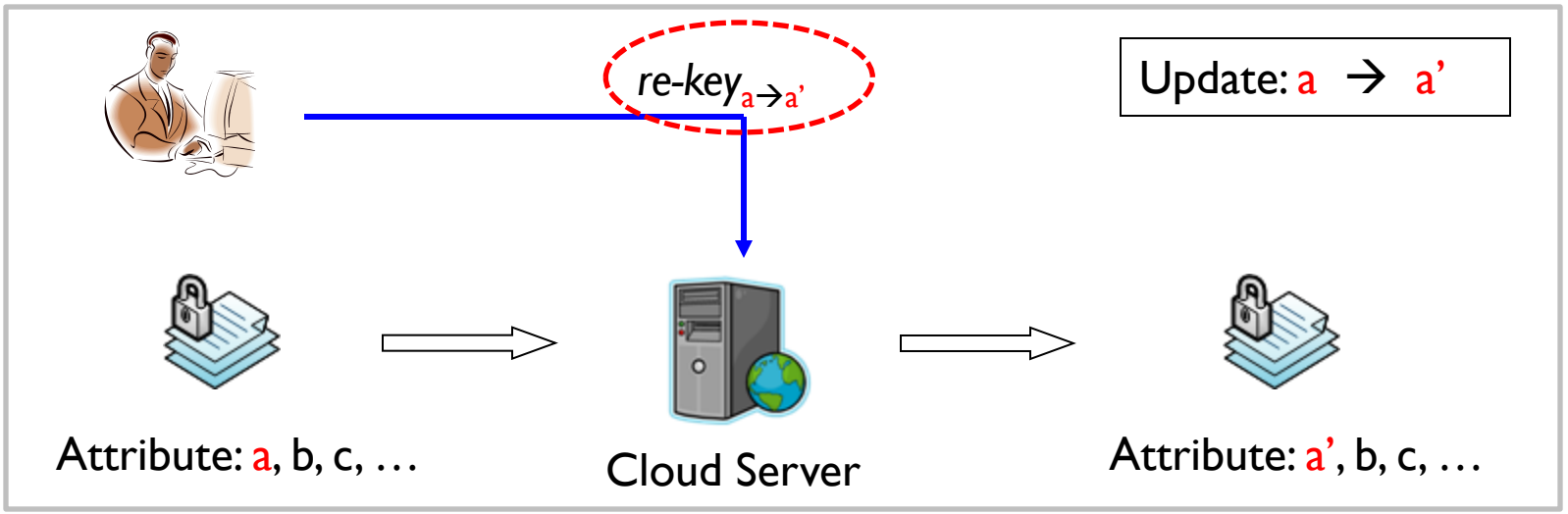
data owner needs to

1) re-encrypt ALL data files that have attribute a.
2) update ALL private keys of non-revoked users ... ntain attribute a

Significant overhead !

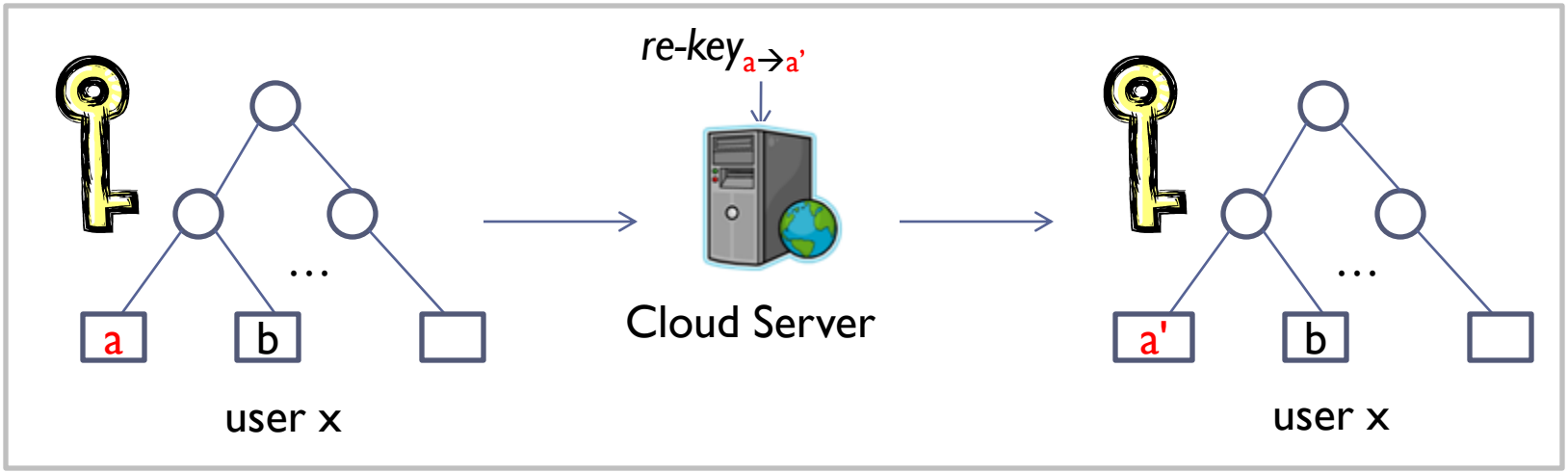
Overhead Reduction on Owner Side

- Delegate file re-encryption to server
  - by proxy re-encryption



Overhead Reduction on Owner Side

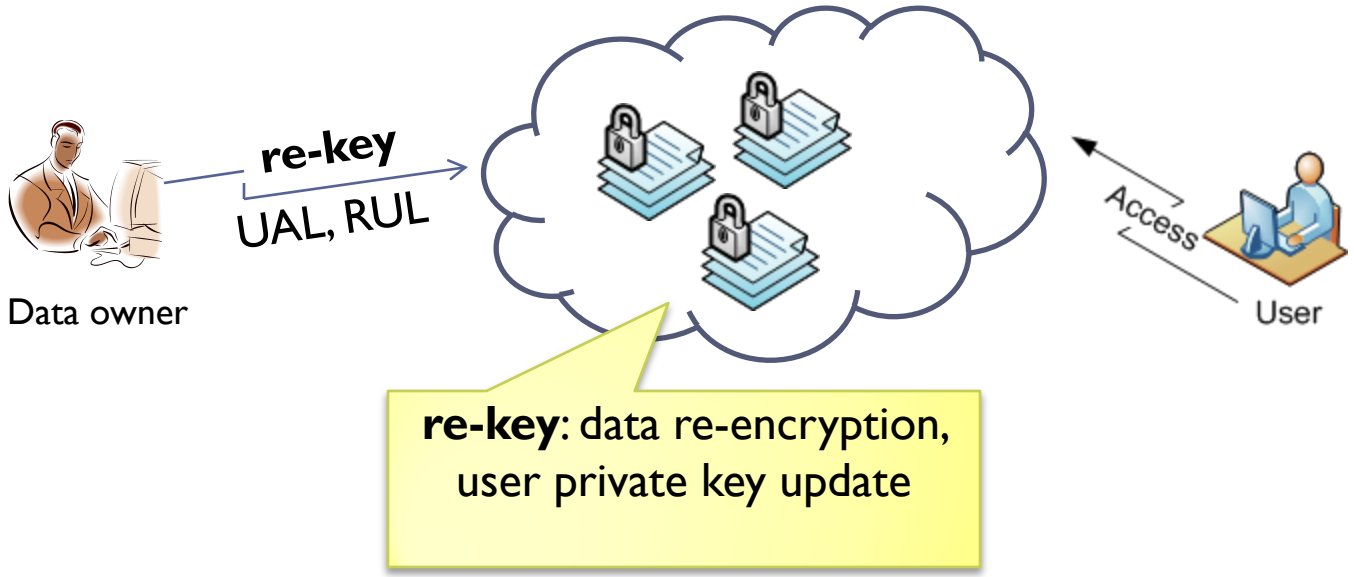
- Delegate user private key update to server
  - By proxy re-encryption



SK: <Sk_a, SK_b, …> → $SK_{a'} = (SK_a)^{a'/a}$ → SK: <$Sk_{a'}$, SK_b, …>

re-key$_{a \to a'}$: a'/a

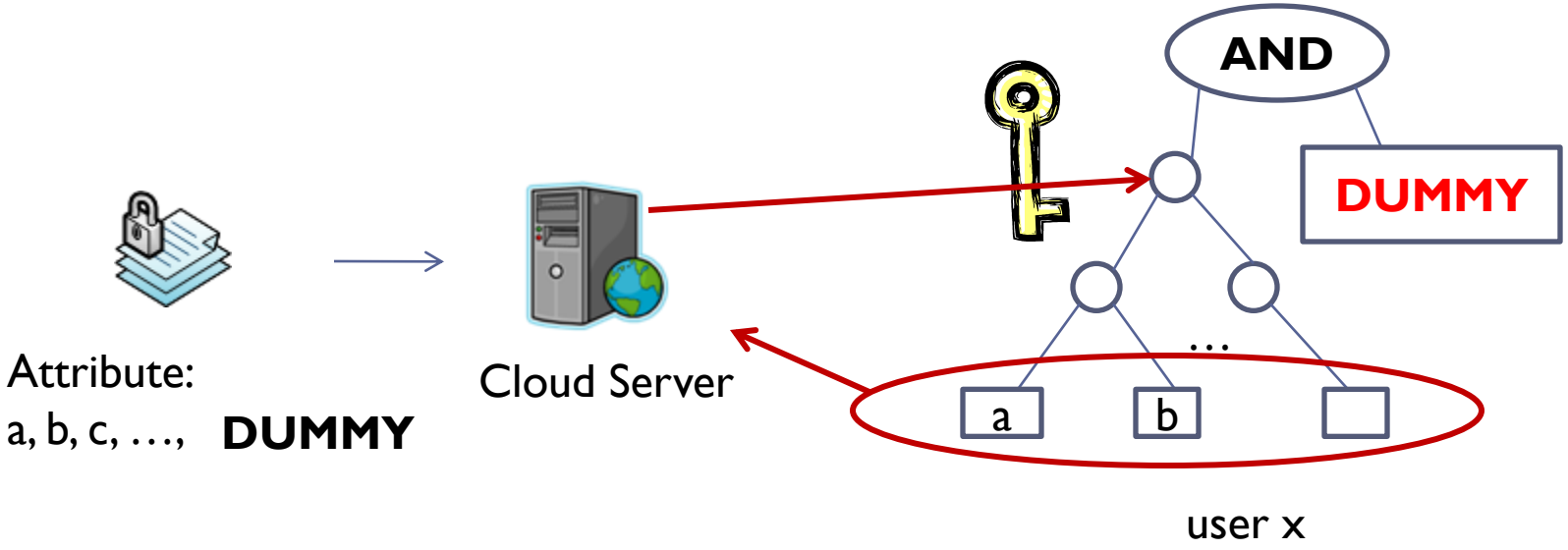
Overhead Reduction on Server Side



- Computation done by server.
- Lazy re-encryption:
  - a user's private key is only updated at the next login
  - a data file is re-encrypted only when it is being accessed.
- Revocation overhead on server is significantly reduced.

Overhead Reduction on User Side

- Decryption delegation
  - Introduce a DUMMY attribute to enable decryption delegation.



- Cloud servers are able to perform the major part of decryption on behalf of the user without knowing the data content.
- Computation on user side is constant.

Cf. [Yu et al. INFOCOM'10]

Comparison of Security and Functionality

	Our scheme	[CCSW09]	[VLDB07]
Fine-grained access control	Yes	Yes	Yes
Privacy guarantee	Yes, Resistant to collusions	Yes	Yes, Not resistant to user-server collusion
Revocation granularity	Attribute-level; User-level; on-demand	N/A	ACL; On-demand
Policy expressiveness	Arbitrary Boolean formula (KP-, CP-ABE), and conjunctive form for MA-ABE	Disjunctive policy	ACL

[VLDB07] di' Vimercati et. al., "Over-encryption: management of access control evolution on outsourced data," in *VLDB '07*.
[CCSW09] Benaloh et.al., "Patient-controlled encryption: ensuring privacy of electronic medical records," in *CCSW '09*.

Comparison of Complexity

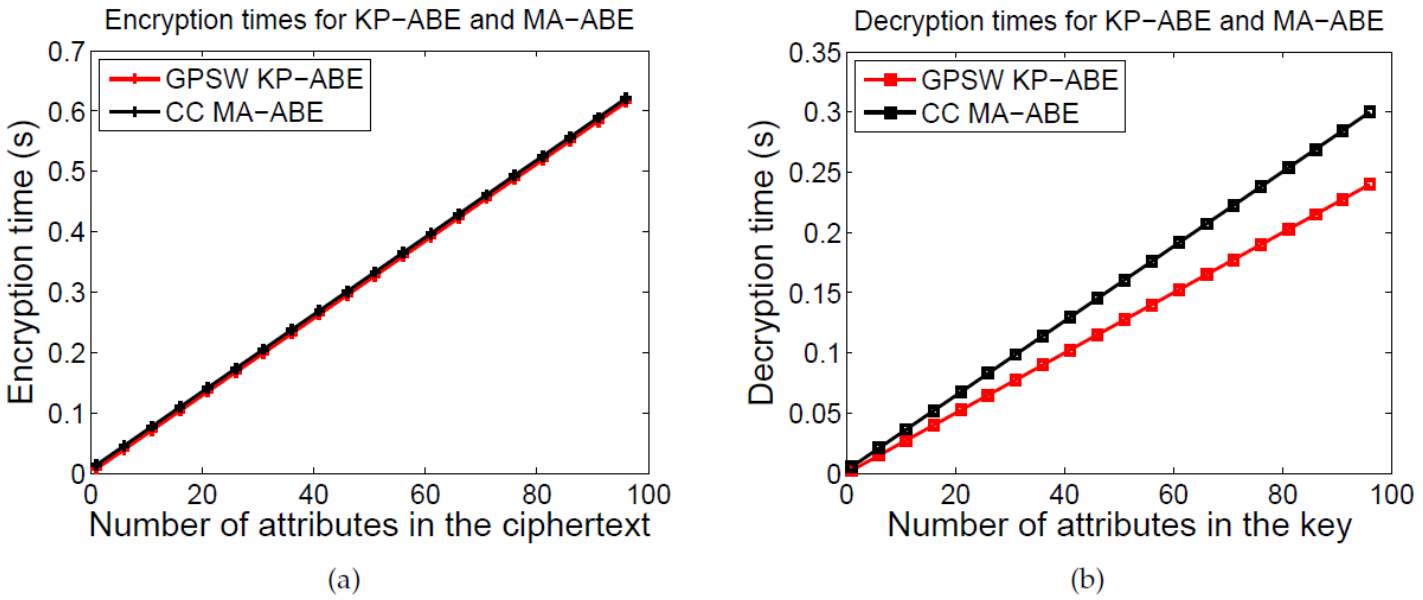
	Our Scheme	[VLDB07]	[CCSW09]
Ciphertext Size	O(tc)	O(1)	O(l)
User Private Key Size	O(tu)	O(No)	O(l*L*No)
Public Key/info. Size	O(|U_D|+|U_R|)	O(No*Nu)	O(No)
Re-keying messages	O(tu)	O(Nu)	N/A

N_o	The number of owners in the system
N_u	The number of data consumers in the system
N_a	Number of users in an attribute group
m	Number of attribute types in the PUD
t_c, t_u	Total number of attributes appeared in CT, sk_u
l	Depth of file hierarchy of an owner's PHR

Implementation/Simulation Results

- Implemented KP-ABE on a 3.4GHz PC

	Pairing (w/ preprocessing)	Exp. in $\mathbb{G}_1$	Exp. in $\mathbb{G}_T$
Time (ms)	2.5	6.4	0.6

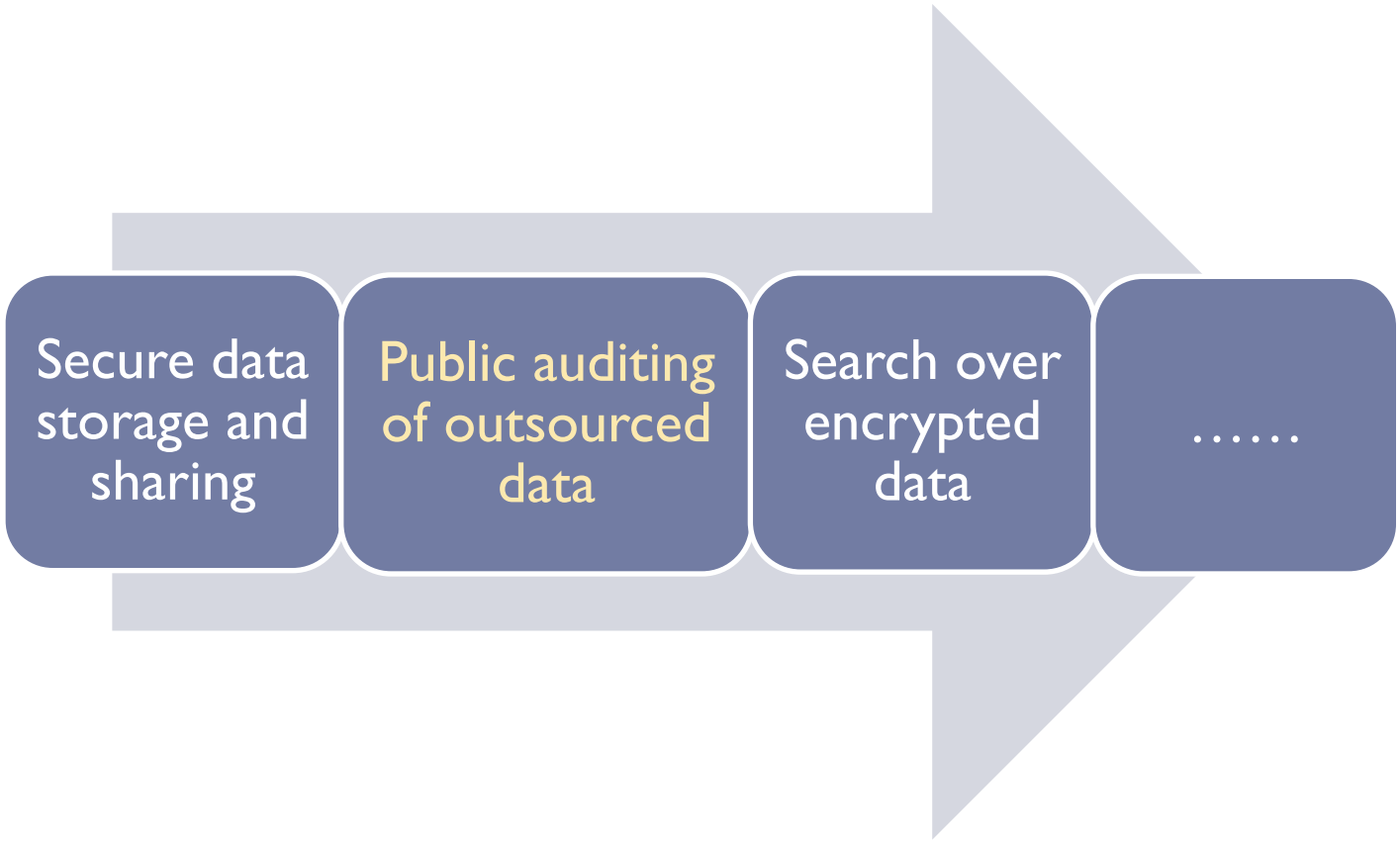


Worst case timing results

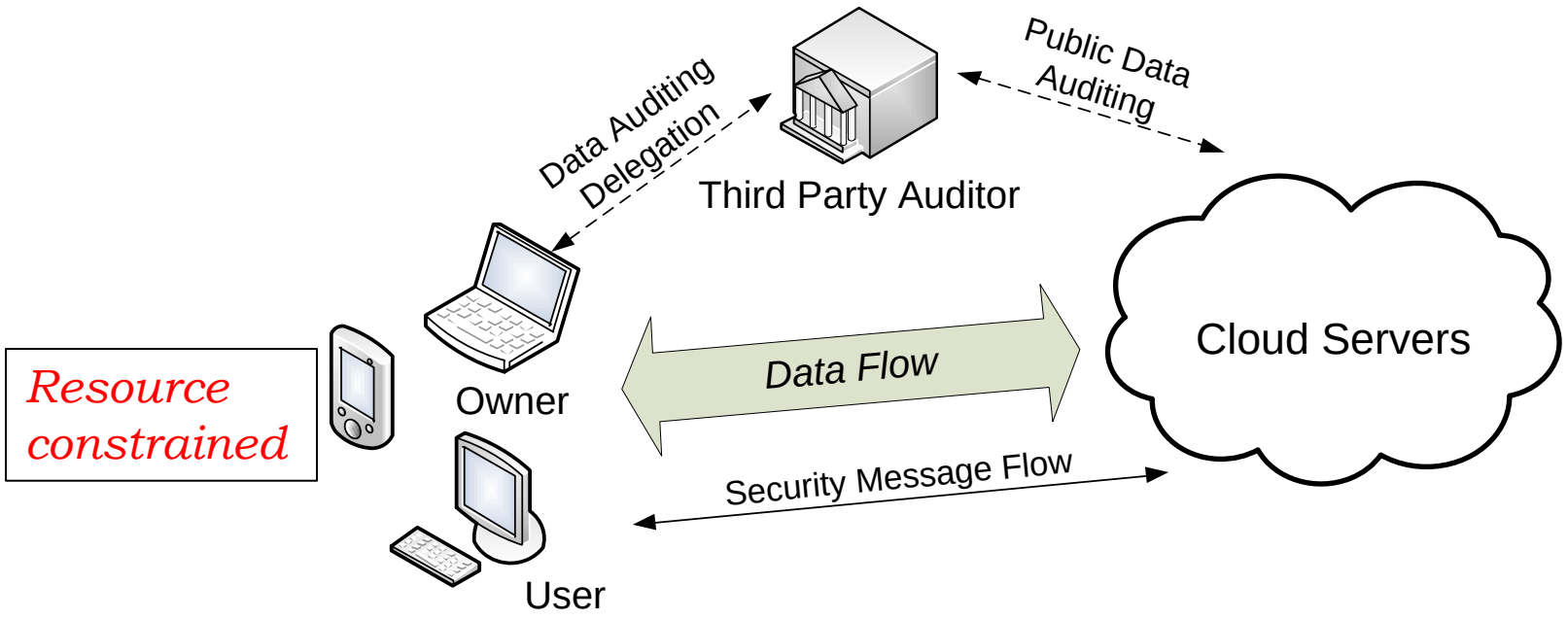
Limitations

- confidentiality to cloud data against cloud server
- Cloud data service is merely a storage
- Encryption and decryption has to be done at the user side.
- Data computation on the server is a challenge
  - With secure cloud server, situation may change
    - Some computation can be done in the server
  - Data at rest still encrypted
- Confidentiality, not integrity yet

Research on Privacy-Assured Data Service



Publicly Auditable Cloud Storage

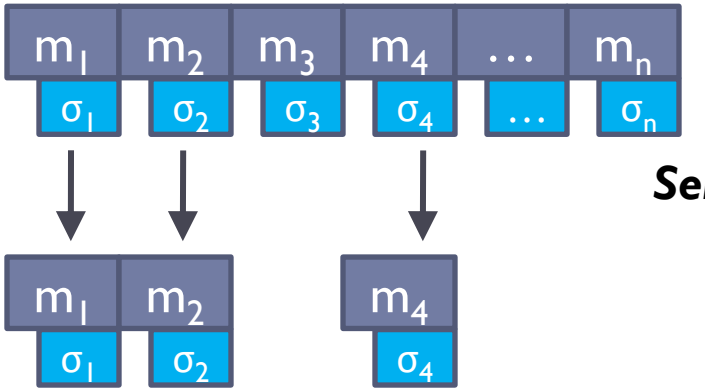


- Data are outsourced to semi-trusted storage server
- Goal: efficient cloud data integrity verification
  - Detect data corrupt or loss, support data dynamics, allow third-party (batch) verification
- Challenge: NO local copy of outsourced data

Straightforward Approaches

- Apply random-sampling approach for probabilistic data integrity guarantee
  - Only check a small portion of the data each time *VS.* retrieve all at once.
  - Can achieve high probabilistic data integrity guarantee

Owner pre-computes an authenticator (e.g., signature) for each file block.



$\sigma_i = MAC_{sk}(i||m_i)$

1. linear bandwidth cost (against sample size),
2. verify block/authenticator pair one-by-one.

Probabilistic Guarantee of Random Sampling

- *Assume r out of n blocks are corrupted, how many blocks should we randomly sample to detect it with high probability?*

- Let X denote the number of corrupted blocks picked by the random-sampling. Then sampling c blocks gives detection probability

$$P = 1 - P\{X = 0\} = 1 - \prod_{i=0}^{c-1}(1 - \min\{\frac{r}{n-i}, 1\})$$

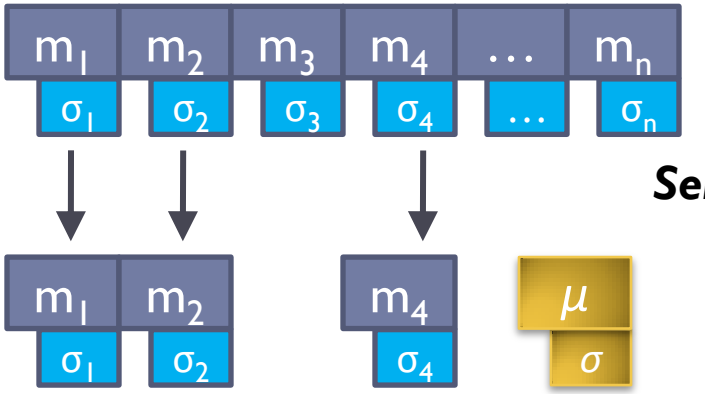
$$\geq 1 - (\frac{n-r}{n})^c = 1 - (1-t)^c, \text{where } t = \frac{r}{n}$$

- If t = 1% of file is corrupted, randomly sample a constant of c = 460 blocks to maintain detection probability P = 0.99.
- Error-correcting code can be used to correct small data errors.

One step forward

- General approach is to employ homomorphic authenticator technique to achieve constant-bandwidth remote data integrity checking.
  - Ateniese et al. CCS 2007
  - Shacham et al. Asiacrypt 2008
  - Wang et al. ESORICS 2009
  - Erway et al. CCS 2009

Owner pre-computes an authenticator for each file block.



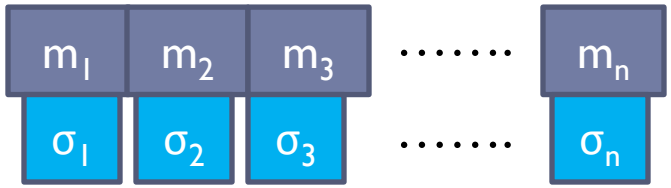
only need to verify μ and σ

small and **constant** bandwidth

blocks and authenticators are combined into single value

When extending to data dynamics

F



$$\sigma_i \leftarrow (h(\nu||i) \cdot g^{m_i})^d$$

G. Ateniese et al. 07
RSA based

$$\sigma_i \leftarrow (H(name||i) \cdot u^{m_i})^x$$

H. Shacham et al. 08
BLS signature based

v, name: randomly chosen labels for file names;
d, x: related private keys; *H*(.), *h*(.) : hash to point functions.

- **Direct extension to data dynamics may have security problems.**
  - E.g., block modification from m_i to $m_i + \Delta m$ allows adversary to obtain Δm and $(u^{\Delta m})^x$ by dividing newly computed σ_i' and original σ_i

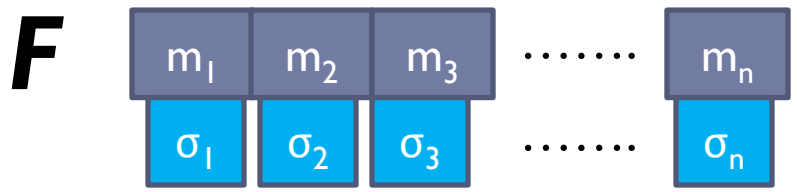
$$\frac{\sigma_i'}{\sigma_i} = \frac{(H(name||i) \cdot u^{m_i + \Delta m})^x}{(H(name||i) \cdot u^{m_i})^x} = (u^{\Delta m})^x$$

  - Adversary could now maliciously modify any block m_s to $m_s^* = m_s + \Delta m$ and forge legitimate authenticator σ_s^* as:

$$\sigma_s^* = \sigma_s \cdot (u^{\Delta m_s})^x = (H(name||s) \cdot u^{m_s})^x \cdot (u^{\Delta m})^x$$
$$= (H(name||s) \cdot u^{m_s + \Delta m})^x$$

- New authenticator construction is required to avoid such attack, where $H(name||i)$ must change for each modification operation.

When extending to data dynamics



$$\sigma_i \leftarrow (h(\nu||i) \cdot g^{m_i})^d$$

G. Ateniese et al. 07
RSA based

$$\sigma_i \leftarrow (H(name||i) \cdot u^{m_i})^x$$

H. Shacham et al. 08
BLS signature based

- **A secure authenticator must enforce the block index/sequence information.**
  - Prevent adversary from using authenticators to obtain proofs for different blocks.
  - E.g., use any valid (m_s, σ_s) pair to pass challenges for corrupted m_t successfully.
- **But keeping index information makes data updates highly inefficient.**
  - E. g., insert a block at any position will affect update (re-computation) on authenticators for all the following blocks.
- Can we eliminate the index information but still enforce block sequence without affecting the security?

Eliminating index info …

- Construct authenticators using $H(m_i)$ instead of $H(name||i)$.

$$\sigma_i \leftarrow (H(m_i) \cdot u^{m_i})^x$$

- New authenticator supports secure block modification operation.
  - $H(m_i)$ changes for every block updates, so the aforementioned attack on block modification is no longer valid.

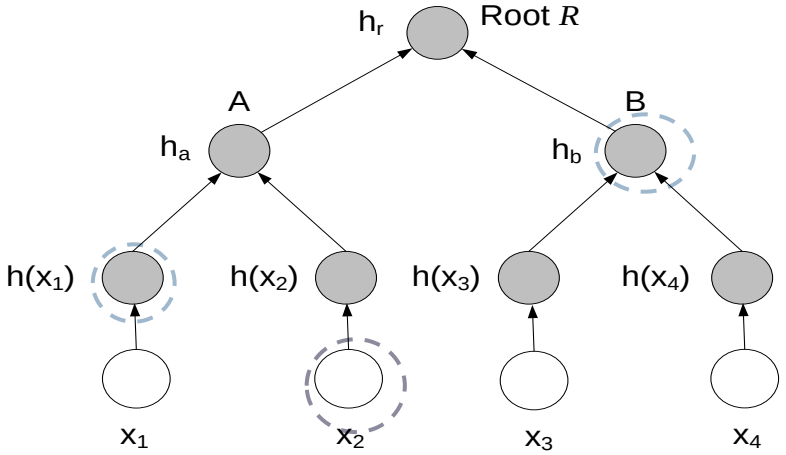
$$\frac{\sigma_i'}{\sigma_i} = \frac{(H(m_i+\Delta m)\cdot u^{m_i+\Delta m})^x}{(H(m_i)\cdot u^{m_i})^x} \not\equiv (u^{\Delta m})^x$$

- Elimination of index for efficient block insertion/deletion operation.
  - When inserting/deleting a block, authenticators for all other blocks remains the same, i.e., no authenticator re-computation is required.

We are yet to need a way to enforce the block index sequence.

Enforce order of data blocks …

- Propose a novel sequence-enforced Merkle Hash Tree (*s*MHT).



Auxiliary Authentication Information (AAI)

To verify the value and position of x_2, we use tree root R and AAI = $\{h(x_1), h_b\}$:

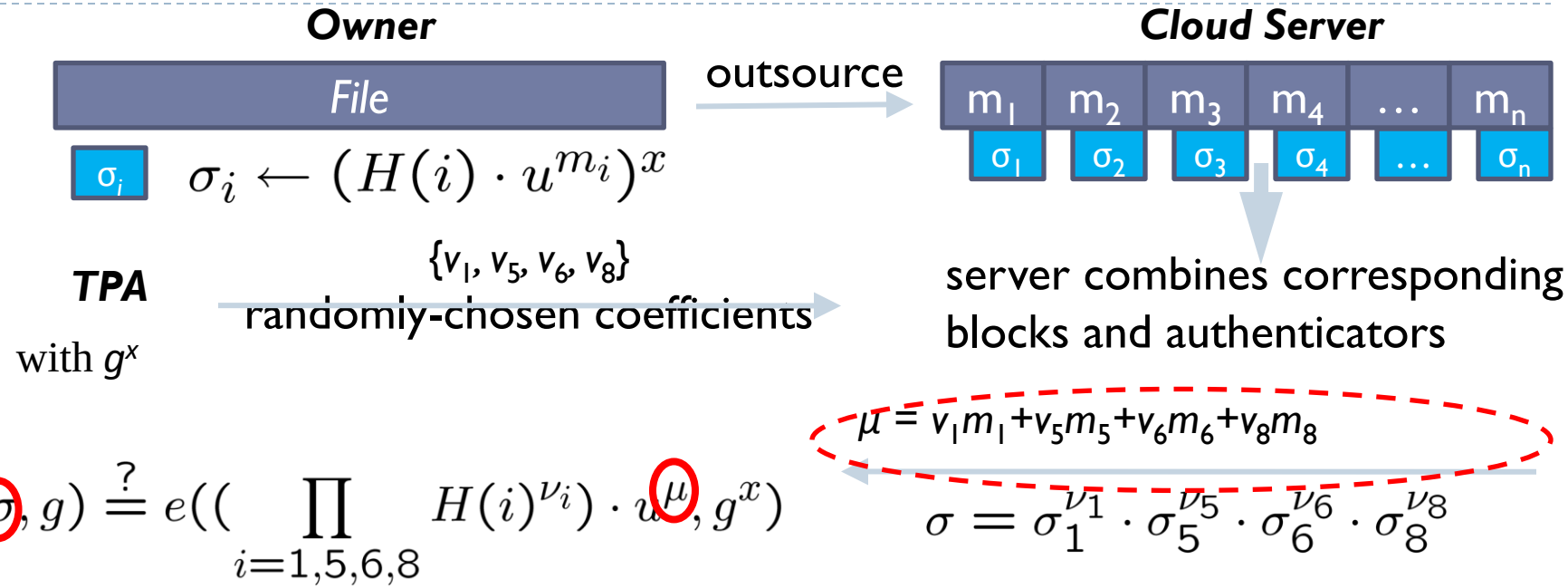
1. Compute $h_a = h(h(x_1) \,||\, h(x_2))$;
2. Verify if $R = h(h_a \,||\, h_b)$.

By following the left-to-right index sequence and the way of computing the tree root, we can uniquely determine both the value and position of any leaf block(s).

- Construct the *s*MHT with an ordered set $\{H(m_i)\}_{i=1,\ldots,n}$ as the leaf nodes, and use root R to ensure the block index information:

$$x_i = H(m_i), i = 1,\ldots, n$$

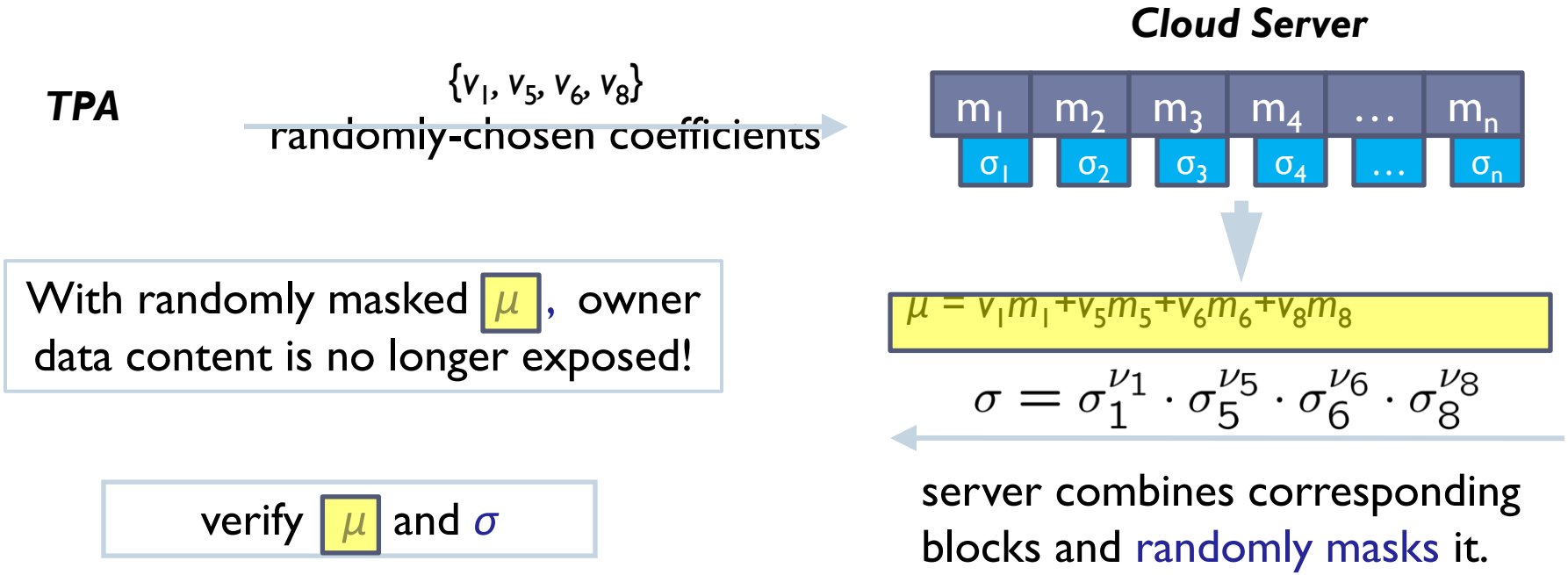
Revisiting Existing Approaches



- $\mu = v_1m_1+v_5m_5+v_6m_6+v_8m_8$ may leak the data to TPA.
  - Direct adoption is unsuitable for public auditing.
- Encrypting data before outsourcing? NOT satisfying.
  - Method not self-contained; Leave the problem to key management.
  - An overkill for unencrypted data (outsourced libraries, scientific data etc.).

Privacy-preserving Public Auditing

- Construct homomorphic aggregation with random masking.
- Achieve privacy-preserving auditing regardless of data encryption.

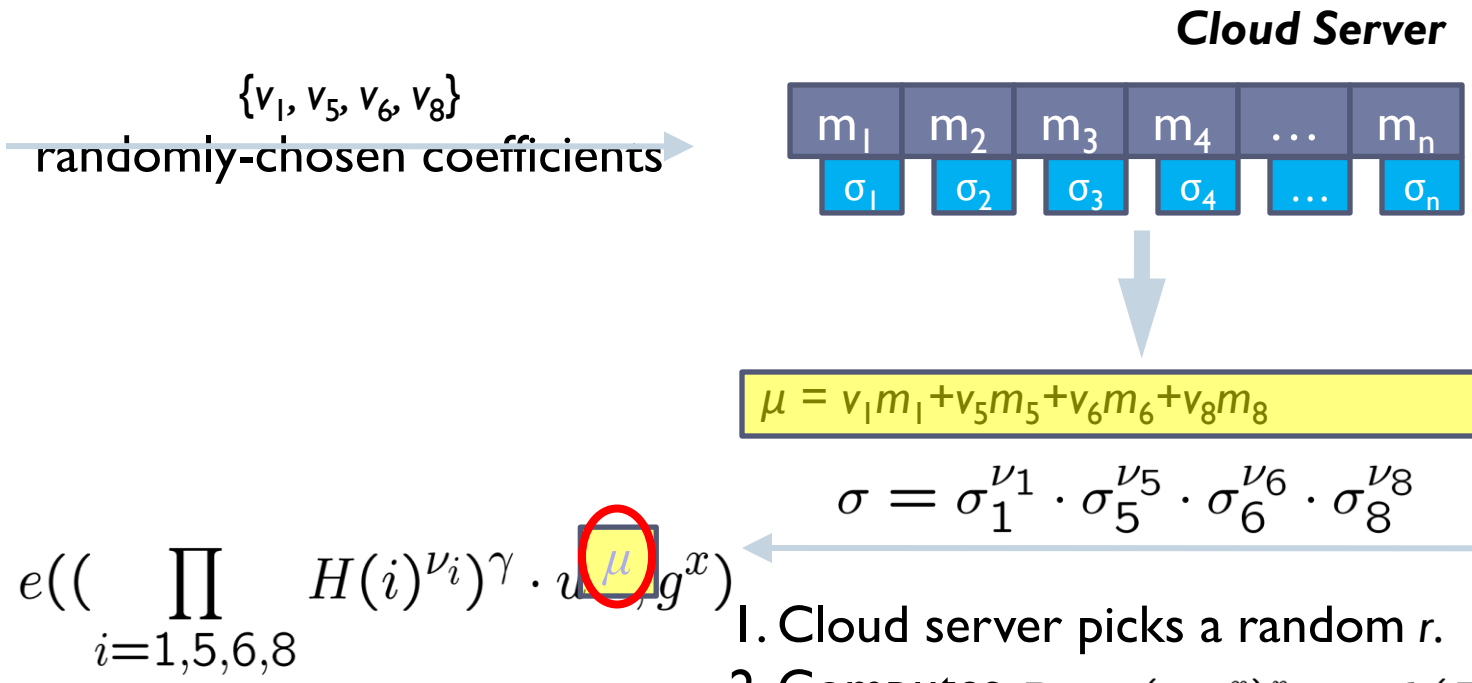


Random masking must not affect storage correctness validation.

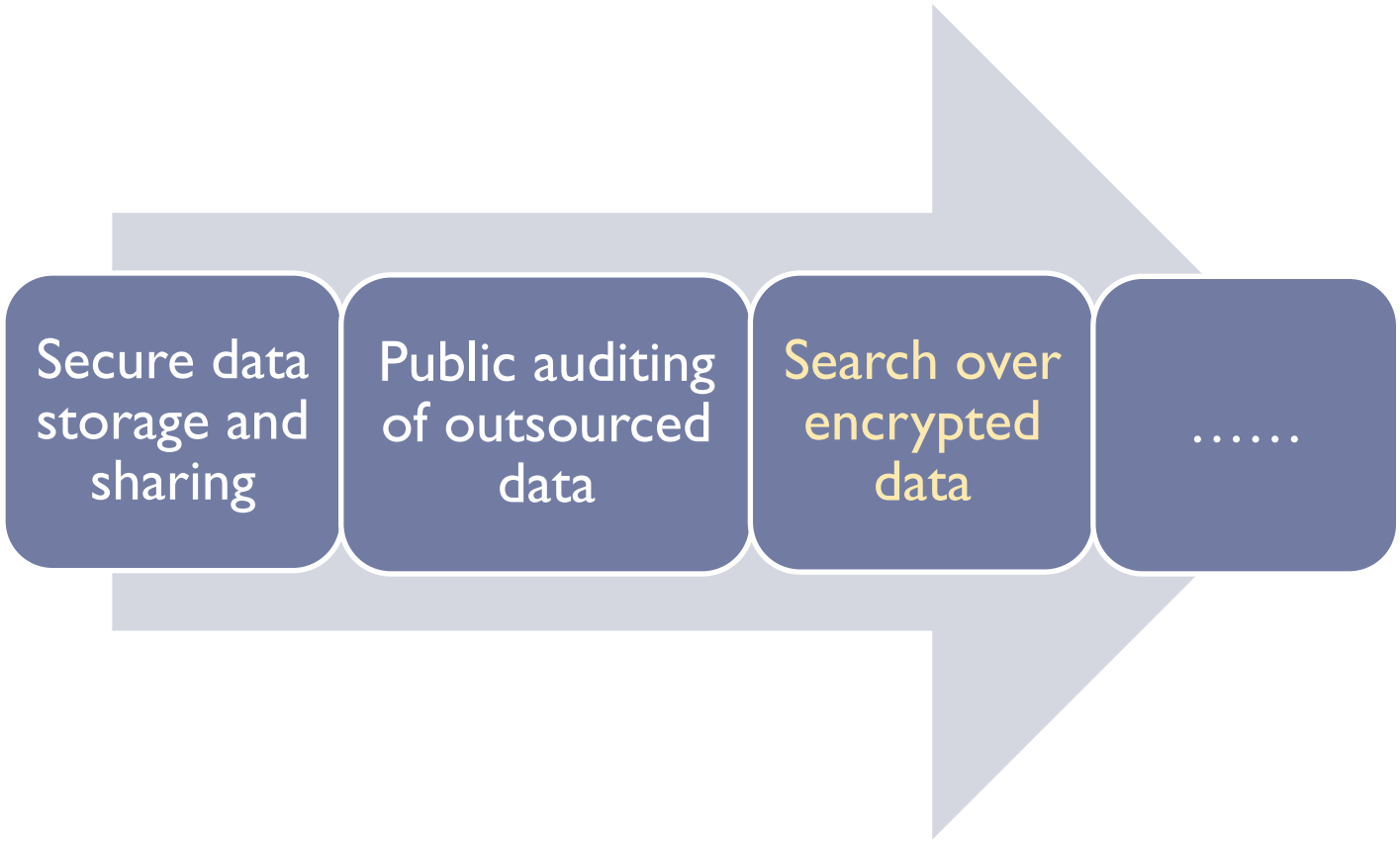
Privacy-preserving Public Auditing

- System Parameters: $(g, g^x, u, e(u, g^x)), h(\cdot) : G_T \rightarrow Z_p.$

$$e : G \times G \rightarrow G_T \quad |G| = |G_T| = p \quad g \in G, < g >= G$$



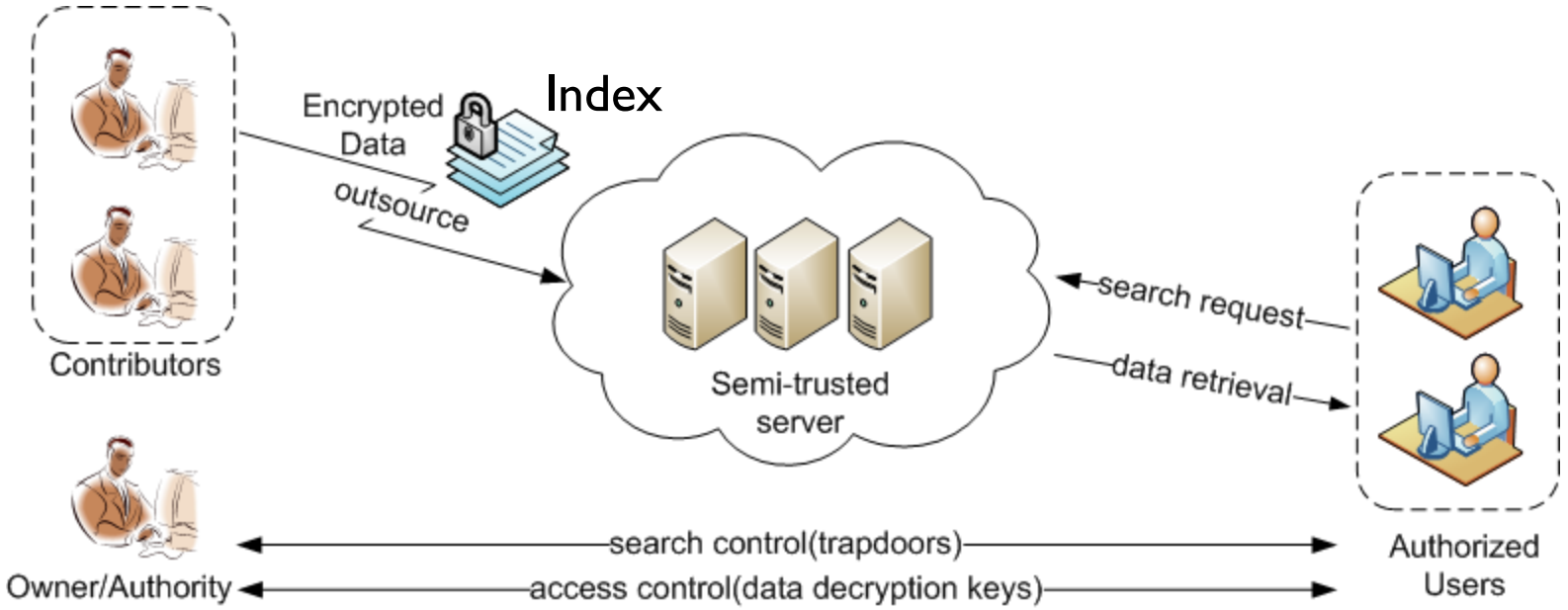
Research on Privacy-Assured Data Service



Privacy-preserving search over encrypted cloud data

Single-owner or multi-owner

multi-user



- Threat Model
  - Cloud server: honest-but-curious (honestly follow the protocol but try to learn private information about user's data); can collude with some users
  - Users: may try to gain unauthorized search access to data; can collude with each other to derive search access beyond their allowed ones

Privacy Concerns

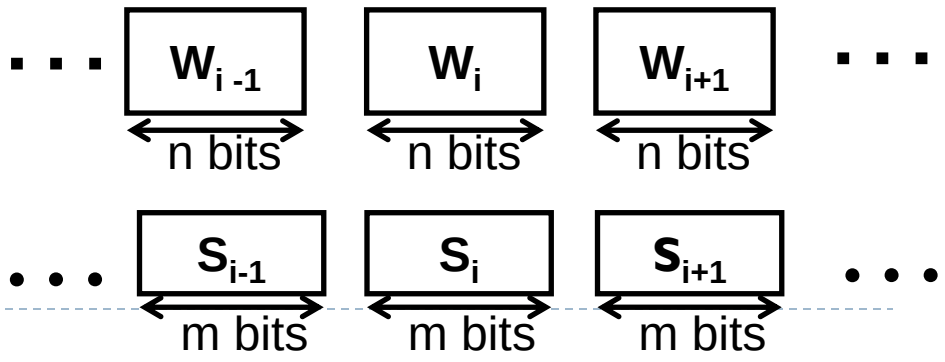
- Data privacy
  - Confidentiality of the outsourced data contents
- Index privacy
  - Confidentiality of data index (if any)
- Search privacy
  - Keyword privacy
    - The keyword in user's query should be hidden
    - Trapdoor is essentially an "encrypted" version of keywords
  - Search pattern privacy
    - A.k.a. trapdoor unlinkability: two queries for the same keyword shall not be linked
    - Requires non-deterministic trapdoor construction
  - Trapdoor unmalleability
    - Should not derive a new trapdoor from existing ones
- Access privacy
  - Hides the access pattern (sequence of returned documents)

All against server

Practical Techniques for Searches on Encrypted Data

D. X. Song, D. Wagner, and A. Perrig **[S&P 2000]**

- **Scenario**: single contributor, outsourced documents
- **Technique overview**: symmetric key, single keyword search
- **Non-Index**: Perform a sequential scan without an index
- **Setup and Notations:**
  - Document: sequence of fixed length words
  - Pseudorandom Generator G and seed:
  - $S \leftarrow G(seed)$, $Si \leftarrow Gi(seed)$
  - Pseudorandom Function F and key K :
  - F_K maps n-m bits to m bits



Practical Techniques for Searches on Encrypted Data (cont'd)

- Pros :
  - Provably secure
    - Derived from security of pseudorandom function/permutation
  - Supporting data dynamics
    - A new document can be simply encrypted and **appended** to the already-stored ciphertext
  - No additional storage

- Cons :
  - Deterministic trapdoor
  - Inefficient: linear scan over encrypted data
  - Support only single fixed-length word search

To do better in efficiency, index-based approach can be adopted!

Index-based Approaches

Secure indexes: E-J. Goh [http://eprint.acr.org/2003/216/]

- Index by document vs. index by keyword
- Build secure index for documents
  - Generate a sub-index for every document consisting of keywords in it
- Techniques used
  - 1. Bloom filters – efficient test for set membership
  - 2. PRF – pseudo-random functions
  - 3. PRG – pseudo-random generator

Secure index Construction

- Keygen(s):

  Given a security parameter s,

  choose a pseudo-random function $f : \{0, 1\}^n \times \{0, 1\}^s \rightarrow \{0, 1\}^s$ and the master key $K_{priv} = (k_1, \ldots, k_r) \leftarrow^R \{0, 1\}^{sr}$.

- Trapdoor(K_{priv},w)

  Given the master key $K_{priv} = (k_1, \ldots, k_r) \in \{0, 1\}^{sr}$ and word w, output the trapdoor for word w as

  $T_w = (f(w, k_1), \ldots, f(w, k_r)) \in \{0, 1\}^{sr}$

Secure index Construction (cont'd)

- BuildIndex(D,K_{priv}):

  a document D comprising of an unique identifier $D_{id} \in \{0, 1\}^n$,

  a list of words $(w_0, \ldots, w_t) \in \{0, 1\}^{nt}$,

  and $K_{priv} = (k_1, \ldots, k_r) \in \{0, 1\}^{sr}$.

  1. For each unique word w_i for $i \in [0, t]$, compute

  (a) the trapdoor: $(x_1 = f(w_i, k_1), \ldots, x_r = f(w_i, k_r))$

  (b) the codeword for w_i in D_{id}:

  $$(y_1 = f(D_{id}, x_1), \ldots, y_r = f(D_{id}, x_r)) \in \{0, 1\}^{sr},$$

  (c) insert $y_1, \ldots, y_r$ into D_{id}'s Bloom filter (BF)

  Non-deterministic "encryption": same keyword is mapped to different bits for different documents

Secure index Construction (cont'd)

- BuildIndex(D,K_{priv}):

2. Compute an upper bound u on the number of words in D.

E.g., extremely assumes one word for every byte in D (after encryption).

3. Blind the index by inserting (u−v)r number of 1's uniformly at random in BF;

v is the number of unique words among the set of t words ($w_0, \ldots, w_t$).

4. Output $ID_{id} = (D_{id}, BF)$ as the index for Did.

Blind to make every BF have similar number of 1s.

Secure index Construction (cont'd)

- Search(T_w, I_D):

  the trapdoor $T_w = (x_1, \ldots, x_r)$ for word w;

  the index $I_{D_{id}} = (D_{id}, BF)$ for document D_{id}.

  1. Compute the codeword for w in D_{id}:

     $(y_1 = f(D_{id}, x_1), \ldots, y_r = f(D_{id}, x_r))$.

  2. Test if BF contains 1's in all r locations denoted by $y_1, \ldots, y_r$.

  3. If so, output 1; Otherwise, output 0.

Secure indexes - Efficiency

- Cost of Algorithms
  - choosing suitable Bloom filter parameters
  - choosing pseudo-random function f
    - keyed hash function HMAC-SHA1: $\{0, 1\}^* \times \{0,1\}^{160} \rightarrow \{0,1\}^{160}$
    - handles arbitrary length word
  - assuming that computing a pseudo-random function is a constant time operation, testing/inserting entries into a Bloom filter are constant time operations
  - Trapdoor algorithm takes O(r) time;
  - The BuildIndex algorithm on a document takes time linear in the number of words in the document;
  - The SearchIndex algorithm takes O(r) time on a document

Secure indexes – Summary

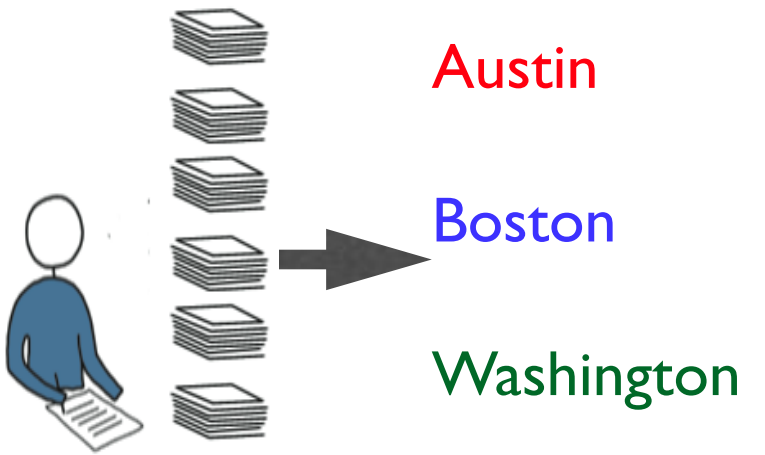
- Pros:
  - Support variable-length words
  - Search efficiency O(Nr)
    - N is the number of files and r is the number of hash functions
  - Dynamics: a new BF can be generated and outsourced to server along with the new document

- Cons:
  - Storage: bloom filter introduces large storage for each file.
  - Weak search semantics: simple boolean keyword search; very hard to control the false positive

Searchable Symmetric Encryption

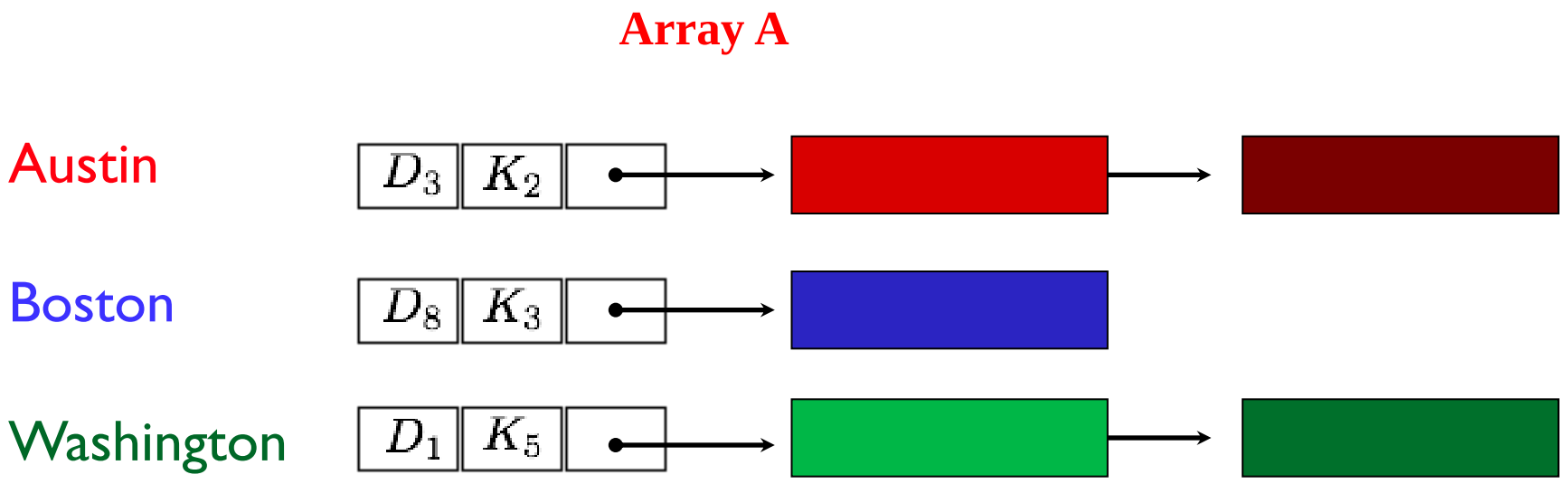
R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky **[CCS 2006]**

- Scenario: single contributor, outsourced documents
- Technique overview: SKC, single keyword search
- Inverted index: an array A and a lookup table T for the whole dataset
  - Build a linked list *Li* for each unique keyword Wi where the elements of the list are the identifiers of all documents which contains Wi



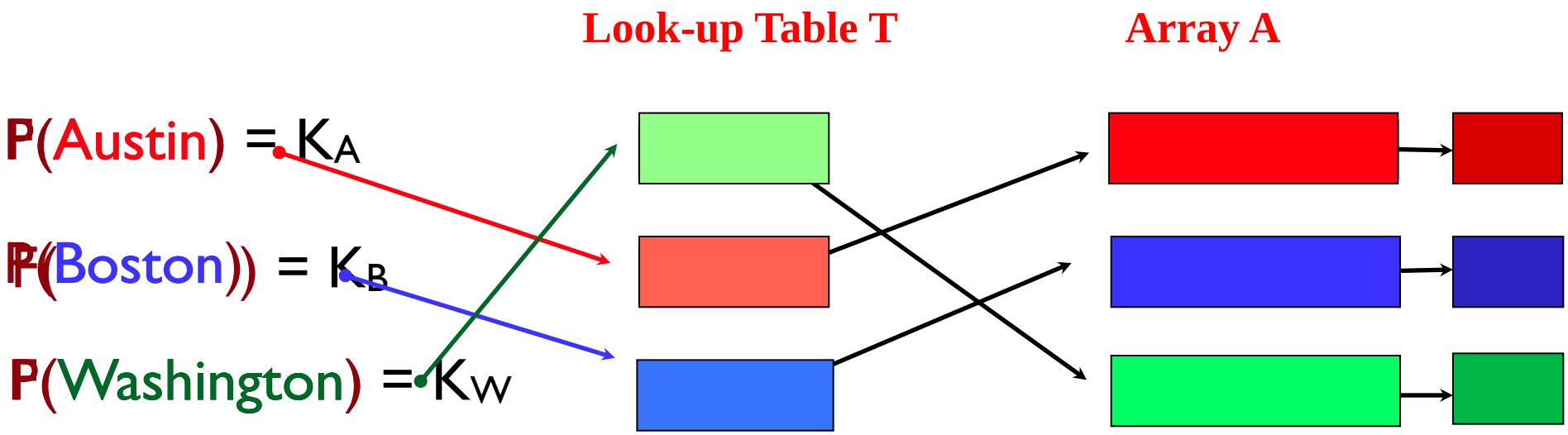
Searchable Symmetric Encryption (cont'd)

- **BuildIndex**: an array A and a lookup table T for the whole dataset
  - Create a linked list for each keyword, consisting IDs of all the documents that contain the keyword.
  - The elements in all linked lists are then encrypted and put into the array A in a mixed order



Searchable Symmetric Encryption (cont'd)

- **BuildIndex**: an array A and a lookup table T for the whole dataset
  - The elements in all linked lists are then encrypted and put into the array A in a mixed order (P: PRP; F: PRF)



Searchable Symmetric Encryption (cont'd)

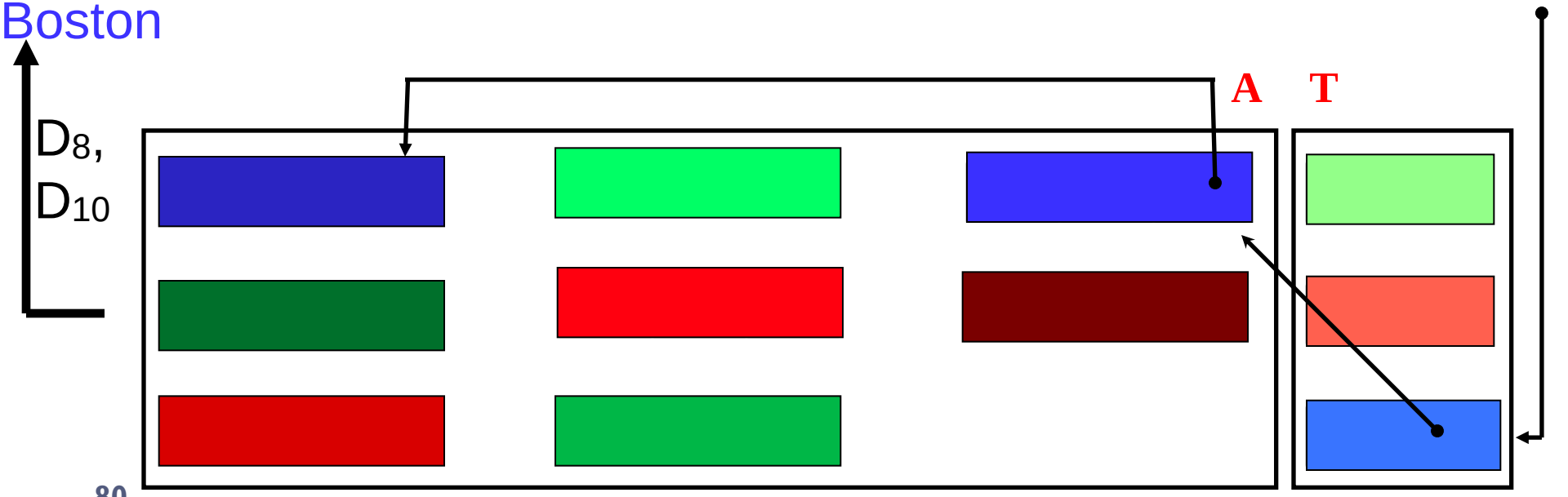
Search :

addr := P(Boston)

key := F(Boston)

Trapdoor := (addr, key)

Boston



Searchable Symmetric Encryption (cont'd)

- Pros:
  - Privacy: hides everything else except search pattern (deterministic trapdoor) and access pattern; provably secure
  - Search Efficiency: server's computation overhead is linear with the number of the documents which contains the querying keyword
  - Storage: better than bloom filter
- Cons:
  - Support only single keyword search
  - Dynamics: to insert a document, the data owner has to retrieve the whole index, decrypt it and build a new index.

Public Key Encryption with Keyword Search

D. Boneh, G. Crescenzo, R. Ostrovsky, and G. Persianoz **[EUROCRYPT 2004]**

- Scenario: multiple contributors, e.g., email gateway
- Technique overview: PKC, bilinear map, BDH(Bilinear Diffie-Hellman Problem)
- Index: a list of encrypted keywords appended as an index per document
  - Contributor of each document encrypts a set of keywords individually with the public key
  - These encrypted keywords are then appended to the encrypted document
- $$\left[E_{A_{pub}}[msg], PEKS(A_{pub}, W_1), \ldots, PEKS(A_{pub}, W_k)\right]$$

(A_{pub} is the public key of the data owner)

Public Key Encryption with Keyword Search

- PEKS Construction

Setup

- two groups G_1, G_2 of prime order p,
- a bilinear map $e: G_1 \times G_1 \to G_2$
  - Bilinear property: for any integer $x, y \in [1, p]$, $e(g^x, g^y) = e(g, g)^{xy}$
- Picking a random $\alpha \in Z_p^*$ and generator g of G_1

$$A_{pub} = [g, h = g^{\alpha}$$

BuildIndex

- Then for keyword W,

$$PEKS(A_{pub}, W) = [g^r, H_2(t)], \text{ where } t = e\left(H_1(W), h^r\right) \in G_2 \text{ for a random } r \in Z_p^*$$

Index privacy: same keyword encrypted differently

Search

- With trapdoor for W,
- Test every PEKS $T_w = H_1(W)^{\alpha} \in G_1$

$$H_2(e(T_w, g^r)) = H_2(t)$$

Public Key Encryption with Keyword Search

- Pros:
  - Multiple contributors
    - Example applications: secure email gateway, e.g., spam filtering
- Cons:
  - Search Inefficiency
    - Linearly scan the appended encrypted keywords in every document, and do pairing computation
  - keyword privacy cannot be guaranteed (common drawback of PKC-based SE)
    - Dictionary attack: Server encrypts every possible keyword with the public key, and then do the search with trapdoor. Based on the search result, server knows the keyword hidden in the trapdoor.

Extending to Ranked Search

- Ranked retrieval are usually more effective than Boolean search.
  - Ranking is **not** supported in existing searchable encryption techniques

Ranking benefits
- Find most/least relevant information quickly
- Avoids unnecessary network traffic

Ranked Search over Encrypted Data

- Targets at → Large scale system usability
  file retrieval accuracy
  user searching experience
  - Retrieve files in a ranked order (top-*k*) according to certain relevance criteria.

- Achieved through frequency based statistical measurement from IR
  - Ranking relevance : term (keyword) frequency × inverse document frequency.

Frequency table with 4 files

	F_1	F_2	F_3	F_4
W_1	3	6	0	0
W_2	2	0	3	5
W_3	5	4	0	1

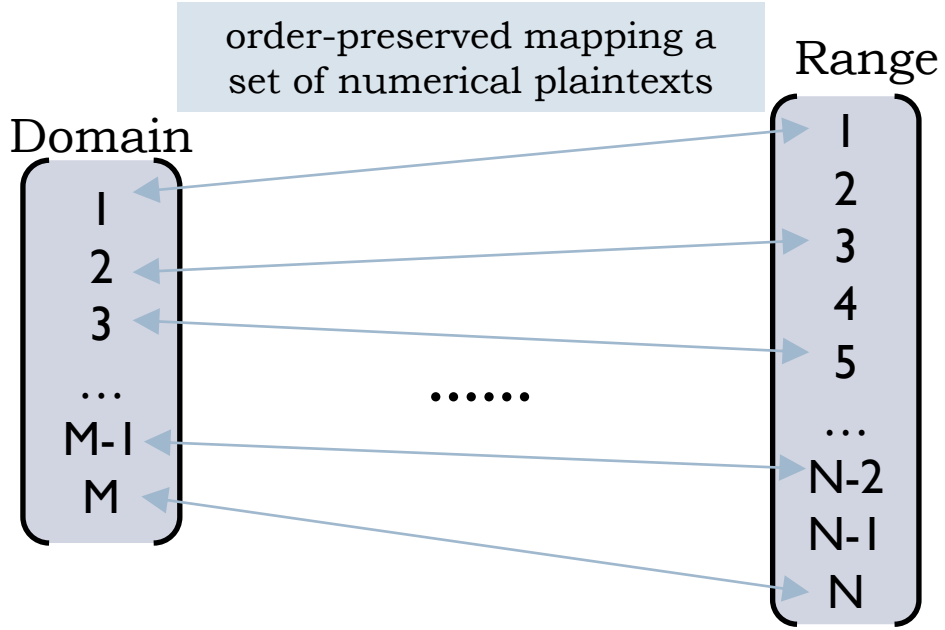
One simple example for TF × IDF rule:

Score $(W_1, F_1) = [3/(3+2+5)] \times \log (4 / 2)$

- For single keyword search: keyword frequency would suffice for ranking.

One Useful Technique: Order-preserving Symmetric Encryption (OPSE)

- OPSE is a deterministic encryption scheme where the numerical ordering of the plaintexts gets preserved.
  - State-of-the-art studied by Boldyreva et al. in Eurocrypt 2009.

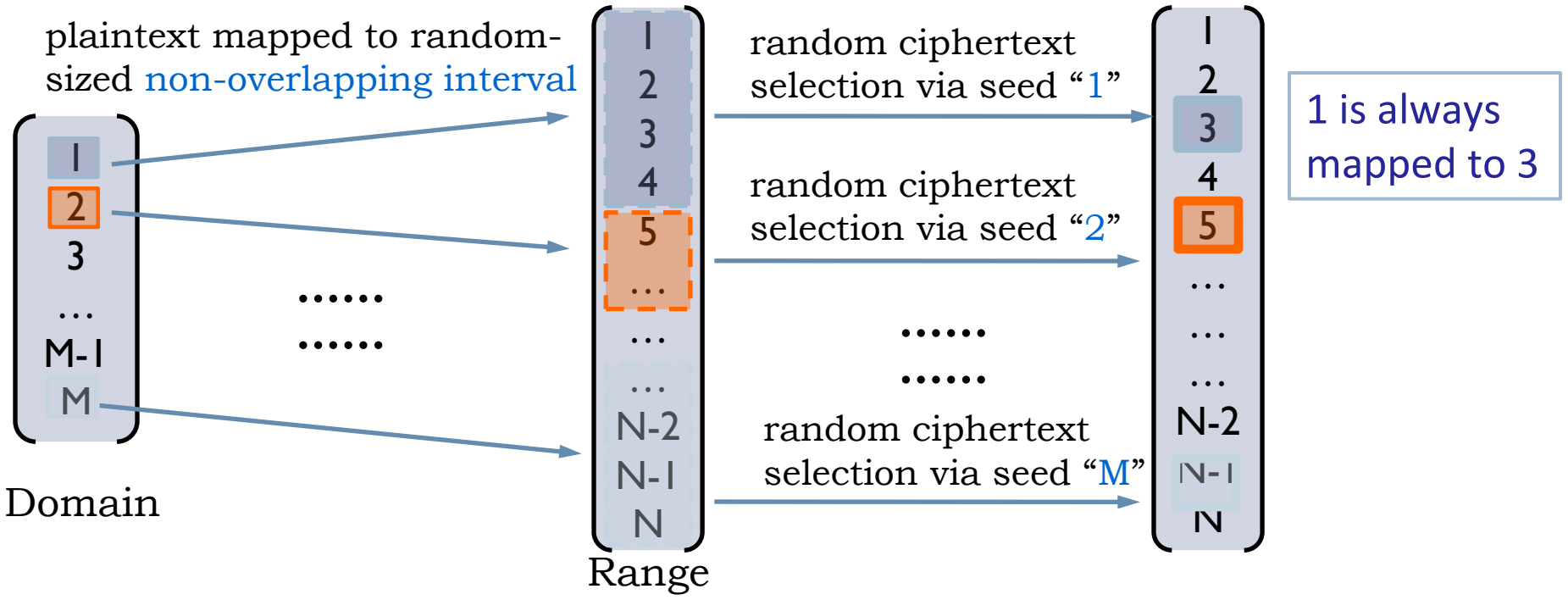


Informally, the order-preserving encryption corresponds to randomly choosing a combination of *M*-out-of-*N* ordered items.

- It's suggested M = N/2 > 80, then # of *M*-out-of-*N* combinations > 2^{80}.
  - We skipped many details of OPSE security definition.

One-to-many Order-preserving Mapping

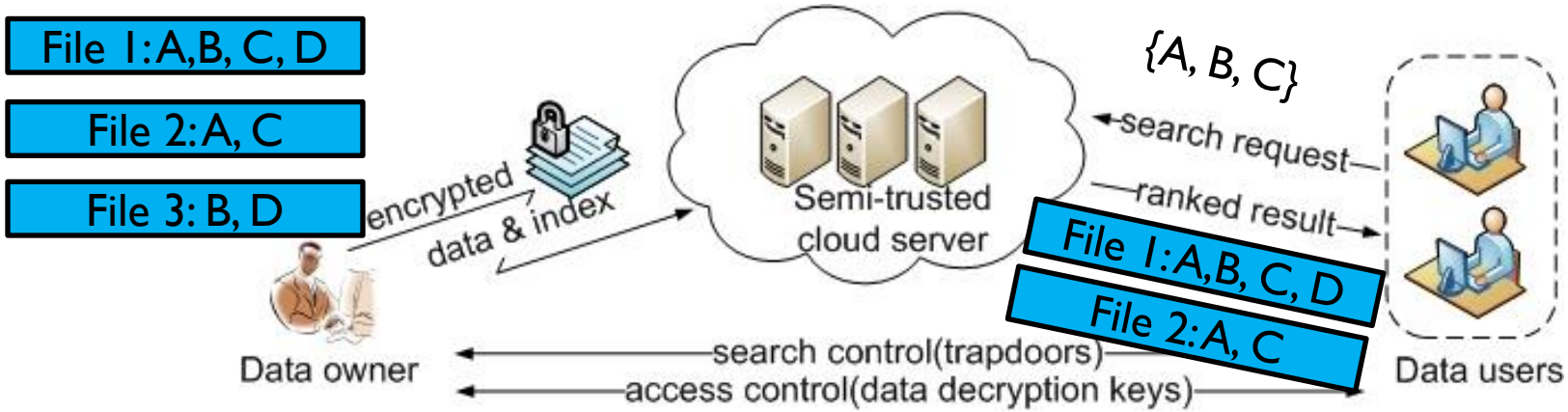
- Brief overview of OPSE plaintext-to-ciphertext assignment process.
  - Plaintext *m* is first mapped to a non-overlapping interval in range, determined by encryption key. Ciphertext *c* is then chosen by using numerical plaintext *m* as seed.



- We propose to further incorporate the unique file IDs $\{F_i\}$ together with the plaintext m's as the final seed for ciphertext selection.

Extending to Multiple-Keyword Ranked Search

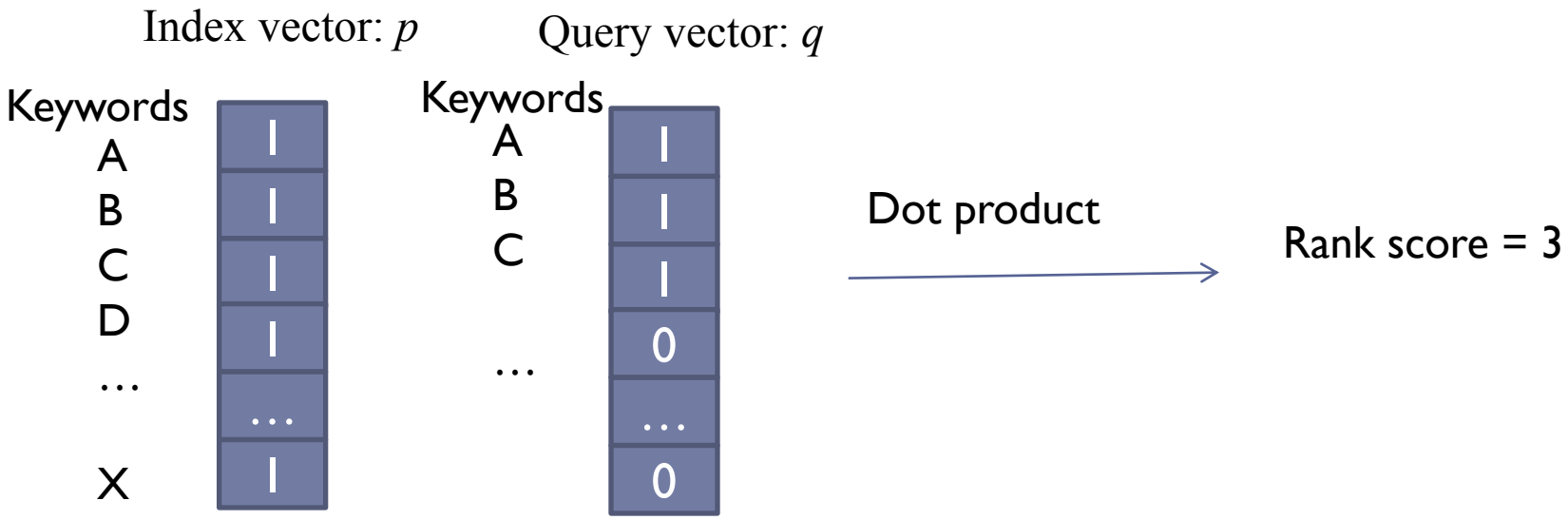
- Typical user's search preference
  - Data users do multiple-keyword search
    - single keyword search often yields far too coarse result
  - Returned documents should be sorted by their similarity with query
    - Documents with more keywords matches will be given higher rank scores



Challenge: how to realize efficient multi-keyword ranked search over encrypted data?

Coordinate matching

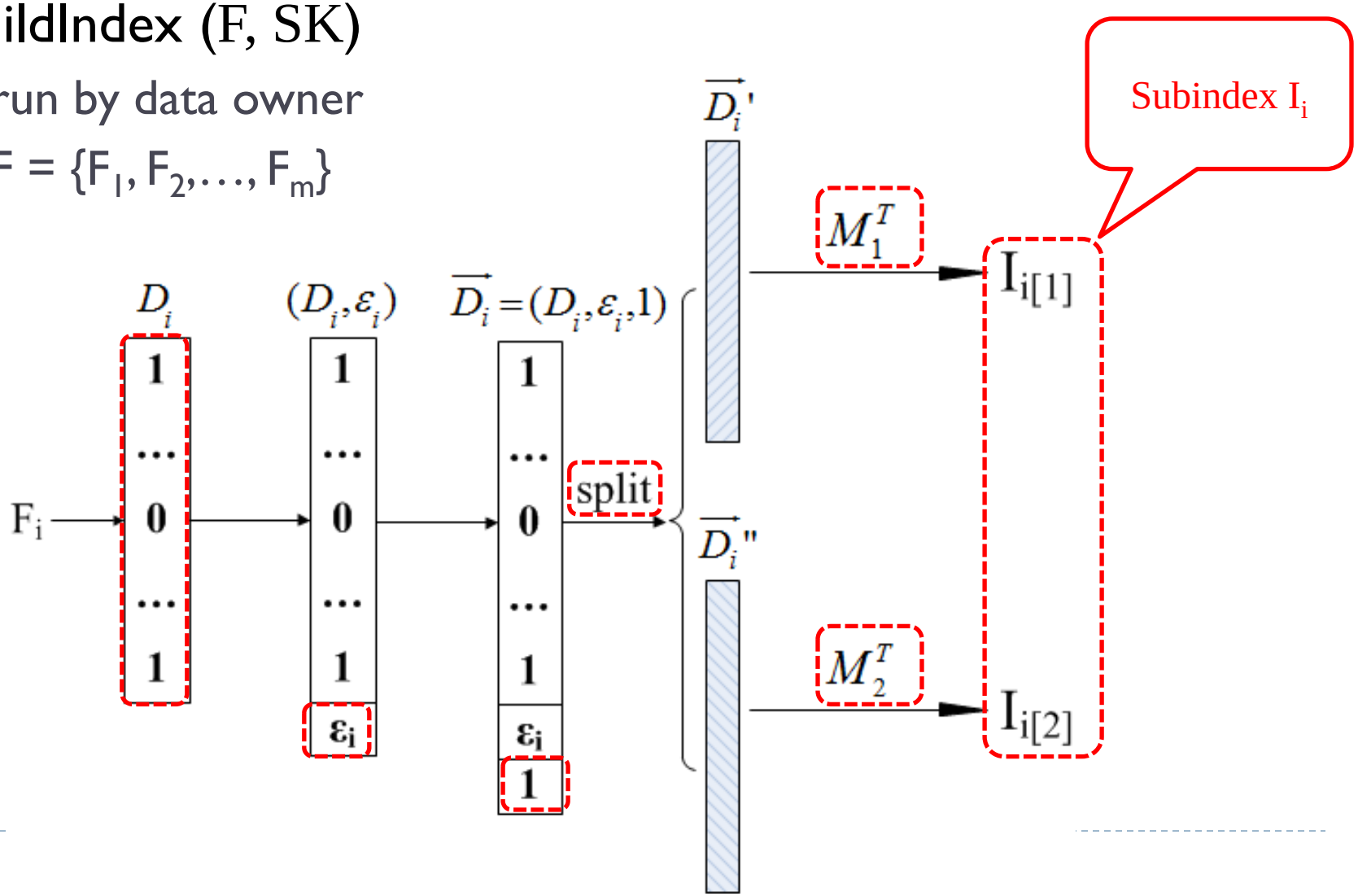
- "Coordinate matching" principle
  - as many matches as possible
  - Widely used metric in IR – information retrieval
- Coordinate matching in plaintext: "Inner Product Similarity"



Question: how to realize coordinate matching over encrypted data, without leaking user privacy?

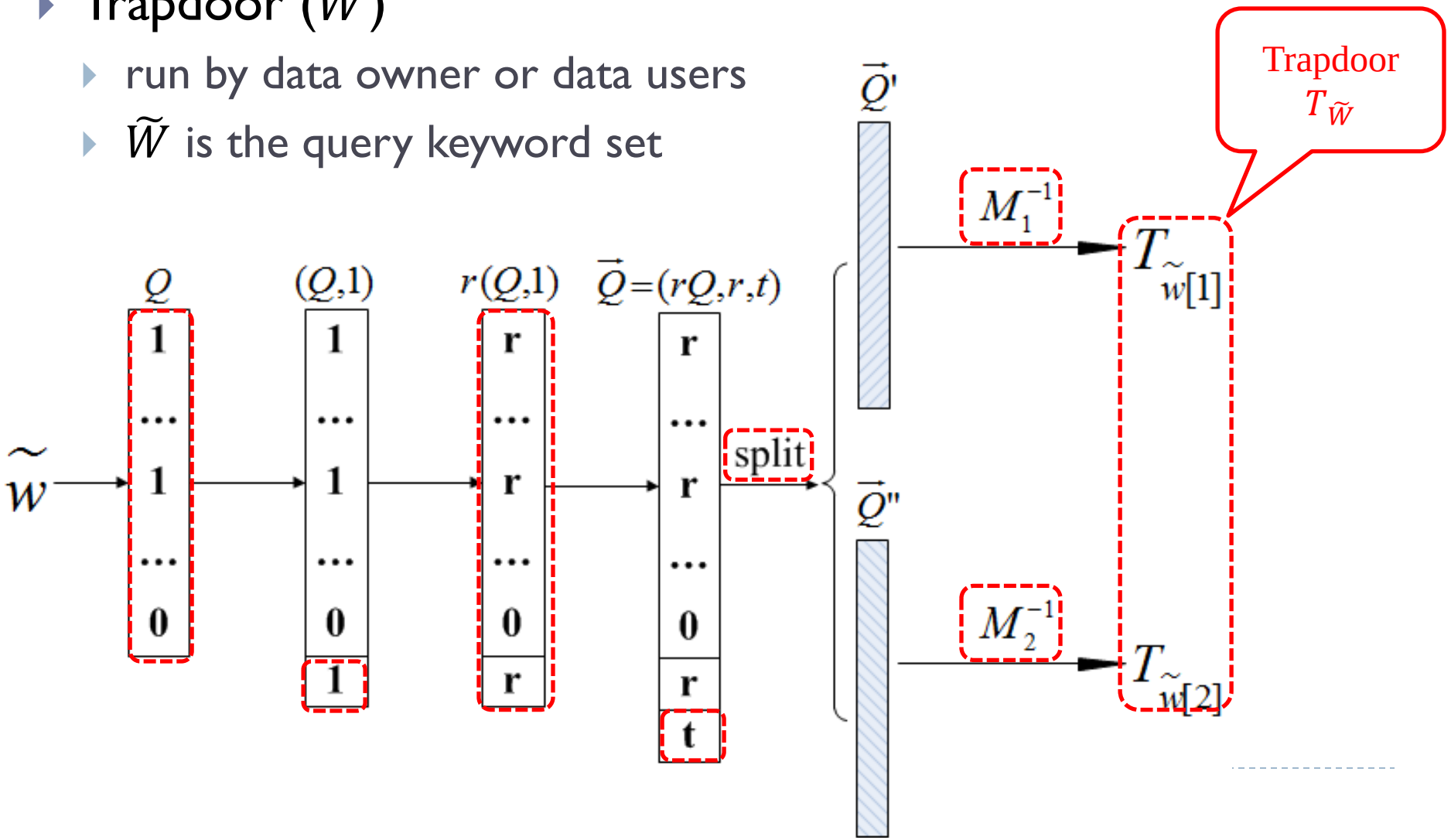
Privacy-Preserving in Known-Ciphertext Model

- BuildIndex (F, SK)
  - run by data owner
  - $F = \{F_1, F_2, \ldots, F_m\}$



Privacy-Preserving in Known-Ciphertext Model

- Trapdoor ($\tilde{W}$)
  - run by data owner or data users
  - $\tilde{W}$ is the query keyword set



Privacy-Preserving in Known-Ciphertext Model

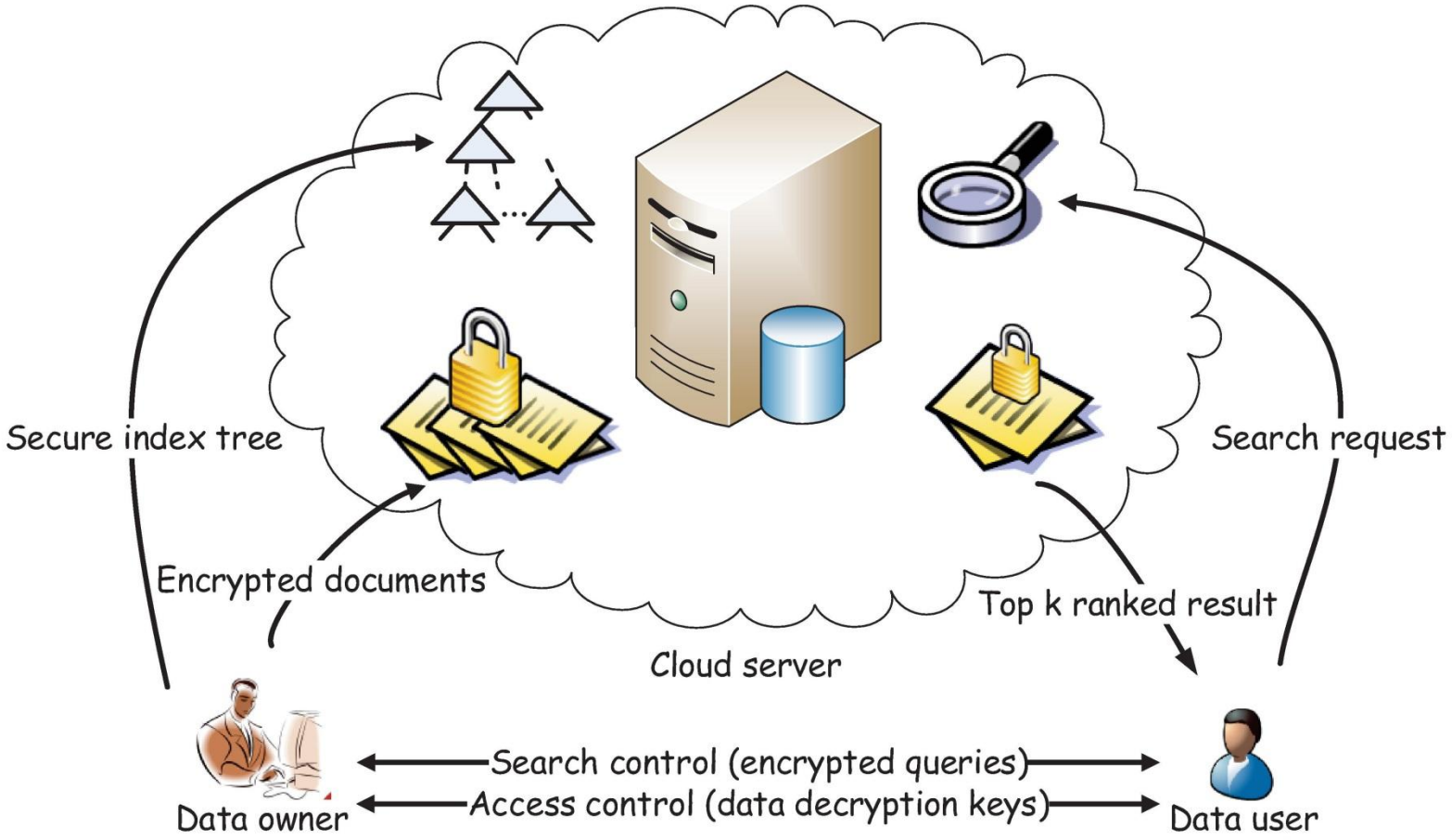
- Query($T_{\widetilde{W}}$, k, I)
  - run by cloud server
  - compute the inner product (similarity score) as $r(D_i \cdot Q+\varepsilon_i)+t$
  - sort all products
  - return top-k ranked document id list

$$
\begin{aligned}
I_i \cdot T_{\widetilde{\mathcal{W}}} &= \{M_1^T \vec{D}_i', M_2^T \vec{D}_i''\} \cdot \{M_1^{-1}\vec{Q}', M_2^{-1}\vec{Q}''\} \\
&= \vec{D}_i' \cdot \vec{Q}\,' + \vec{D}_i'' \cdot \vec{Q}\,'' \\
&= \vec{D}_i \cdot \vec{Q} \\
&= (D_i, \varepsilon_i, 1) \cdot (rQ, r, t) \\
&= r(D_i \cdot Q + \varepsilon_i) + t.
\end{aligned}
$$

Privacy-Preserving in Known-Ciphertext Model

- Functionality
  - ε has a great impact on the search accuracy
    - $\varepsilon \sim N(\mu, \sigma^2)$
    - σ influences the order of similarity scores
- Efficiency
  - In BuildIndex or Trapdoor, two multiplications of a (n + 2) X (n + 2) matrix and a (n + 2)-dimensional vector are involved to generate each subindex or trapdoor
  - In Query, the final similarity score is computed through two multiplications of two (n + 2)-dimensional vectors

Similarity-based Multiple keyword ranking search

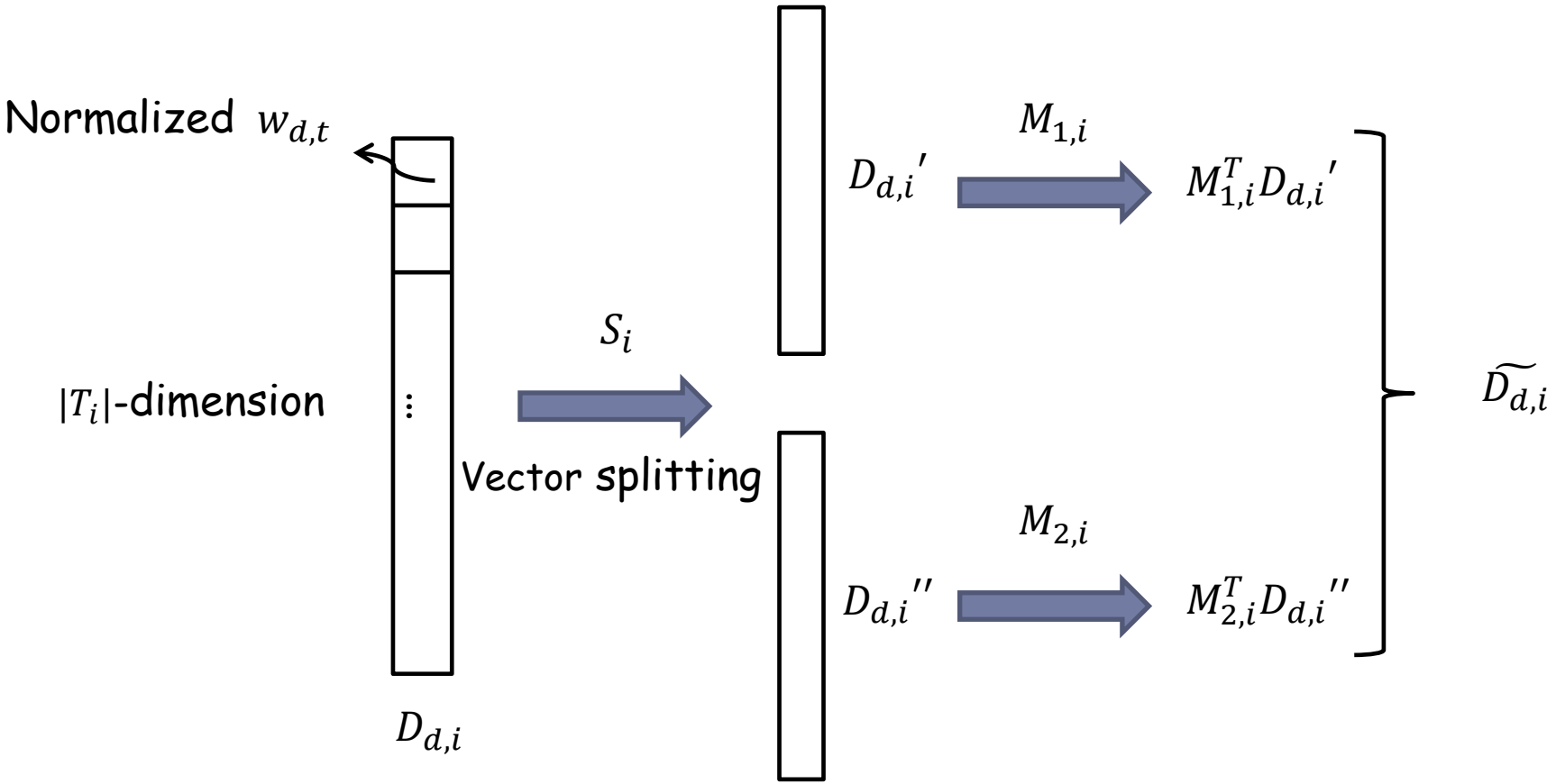


Preliminaries

- Similarity evaluation function
  - $Cos(D_d, Q) = \frac{1}{w_d w_q} \sum_{t \in Q \cap D_d} w_{d,t} \cdot w_{q,t}$
  - where $w_{d,t} = 1 + \ln(f_{d,t})$, $w_{q,t} = \ln\left(1 + \frac{N}{f_t}\right)$, $w_d = \left(\sum_{t=t_1}^{t_n} w^2_{d,t}\right)^{1/2}$, $w_q = \left(\sum_{t=t_1}^{t_n} w^2_{q,t}\right)^{1/2}$
  - $D_d = (w_{d,t_1}, w_{d,t_2}, \ldots, w_{d,t_n})/w_d$
  - $Q = (w_{q,t_1}, w_{q,t_2}, \ldots, w_{q,t_n})/w_q$
  - $0 \leq Cos(D_d, Q) < 1$

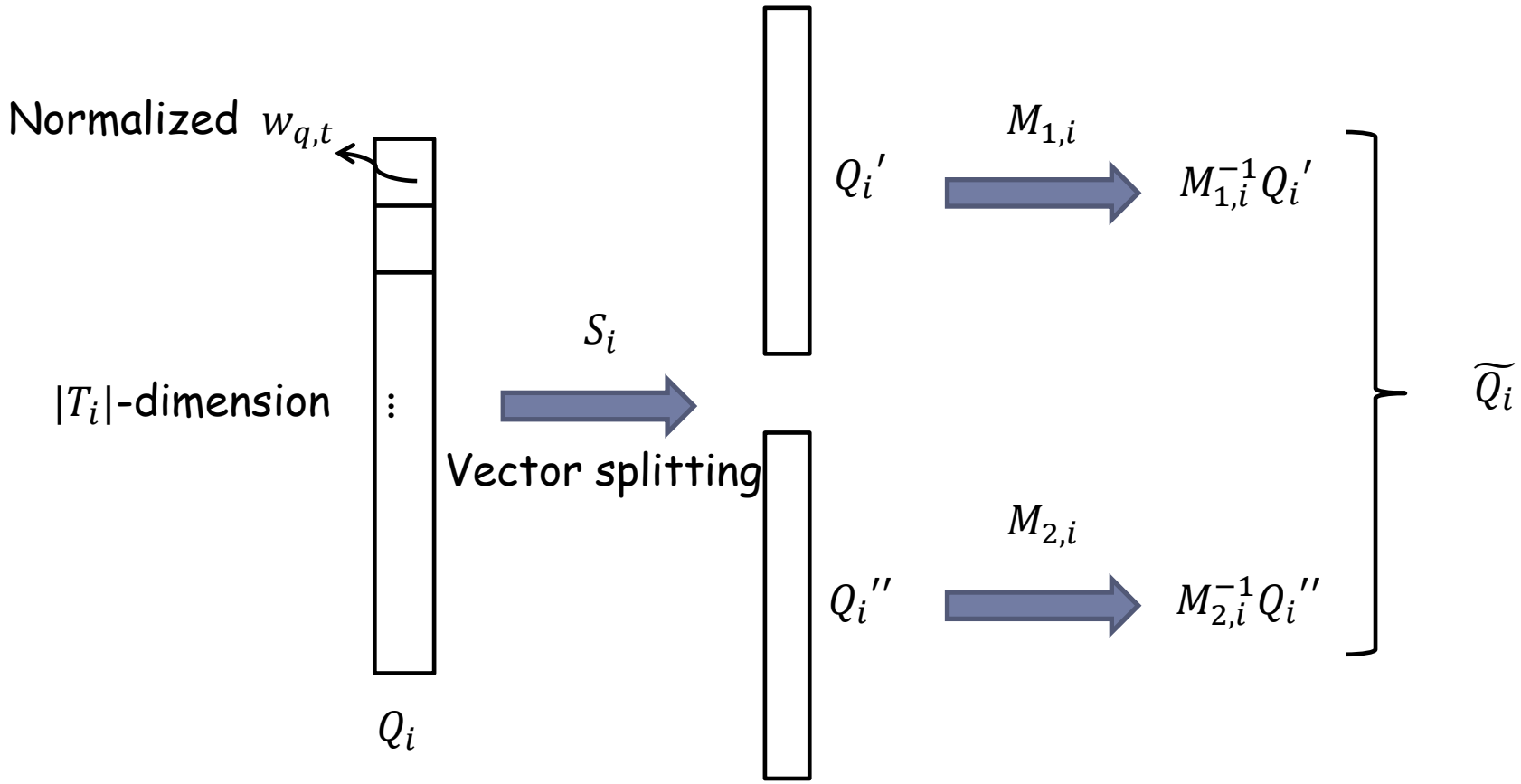
Secure Index Scheme

- GenIndex



Secure Index Scheme

- GenQuery



Secure Index Scheme

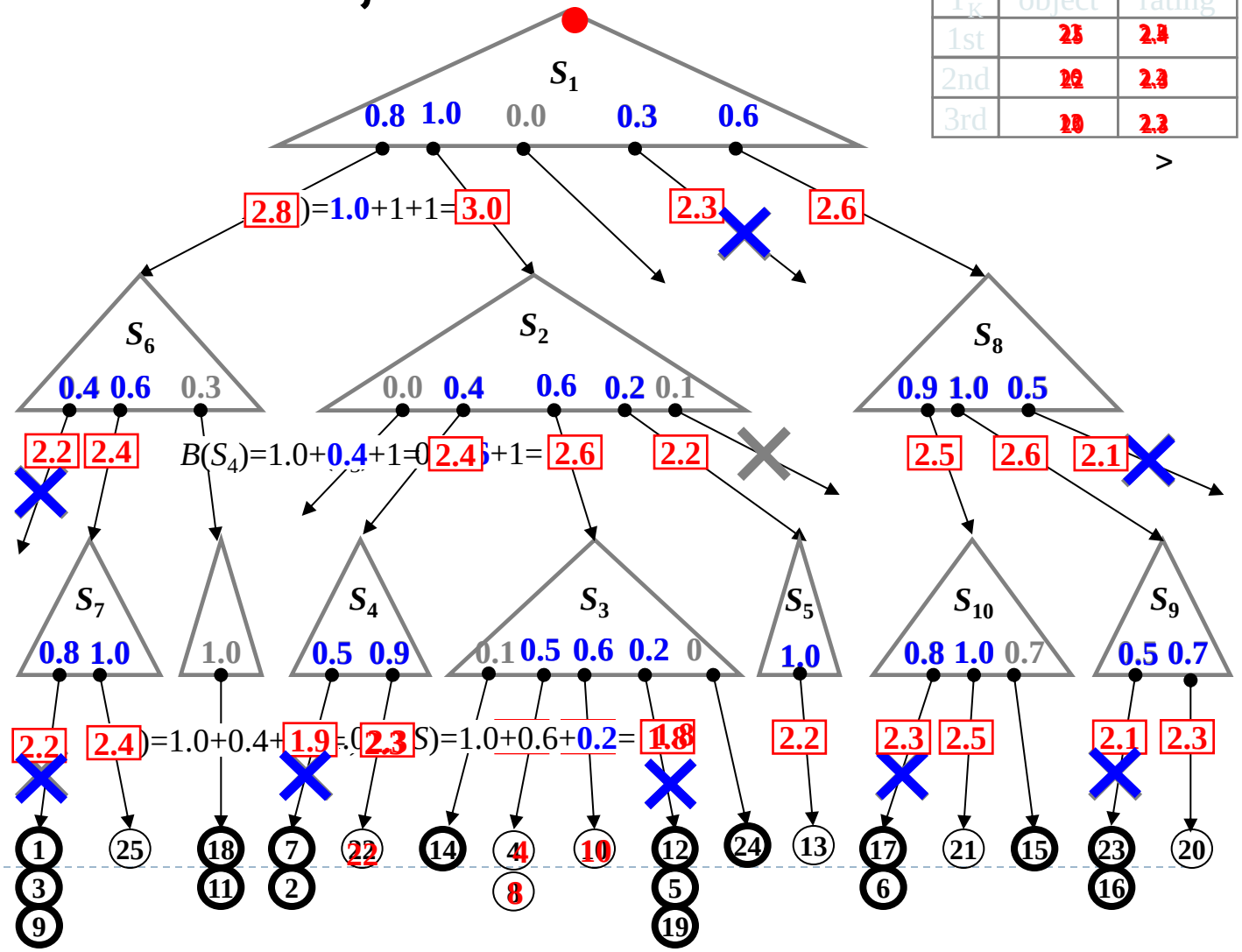
- SimEvaluation:
  - $cosine\left(\widetilde{D_{d,i}}, \widetilde{Q_i}\right)$
    $= \left\{M_{1,i}^T D_{d,i}{}', M_{2,i}^T D_{d,i}{}''\right\} \cdot \{M_{1,i}^{-1} Q_i{}', M_{2,i}^{-1} Q_i{}''\} = D_{d,i} \cdot Q_i$

- The final similarity score for document d is:

  - $\sum_{i=1}^{h} D_{d,i} \cdot Q_i = D_d \cdot Q$

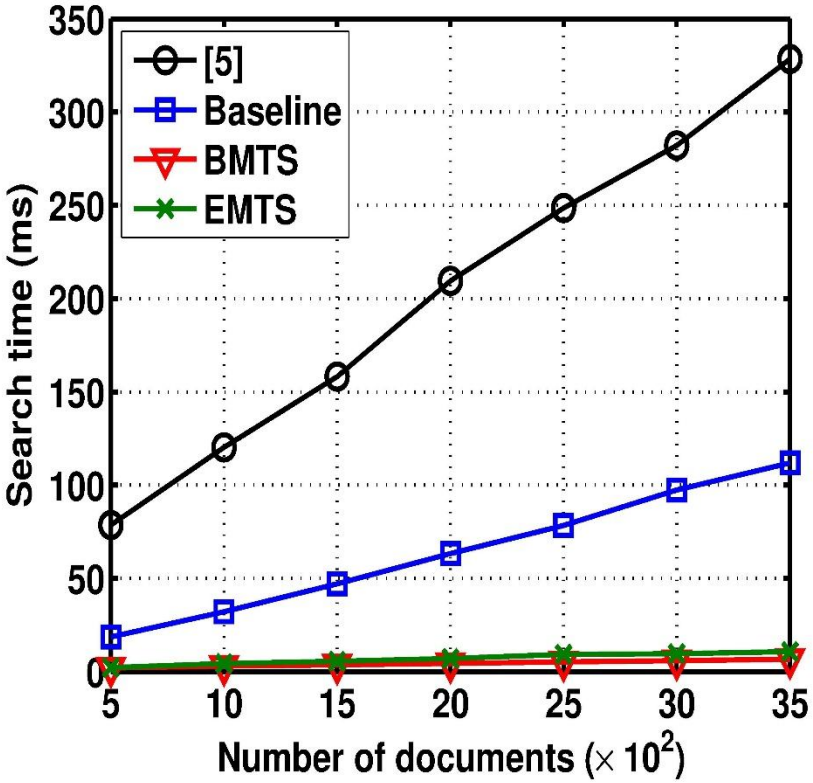
Multi-dimensional Algorithm Based Search

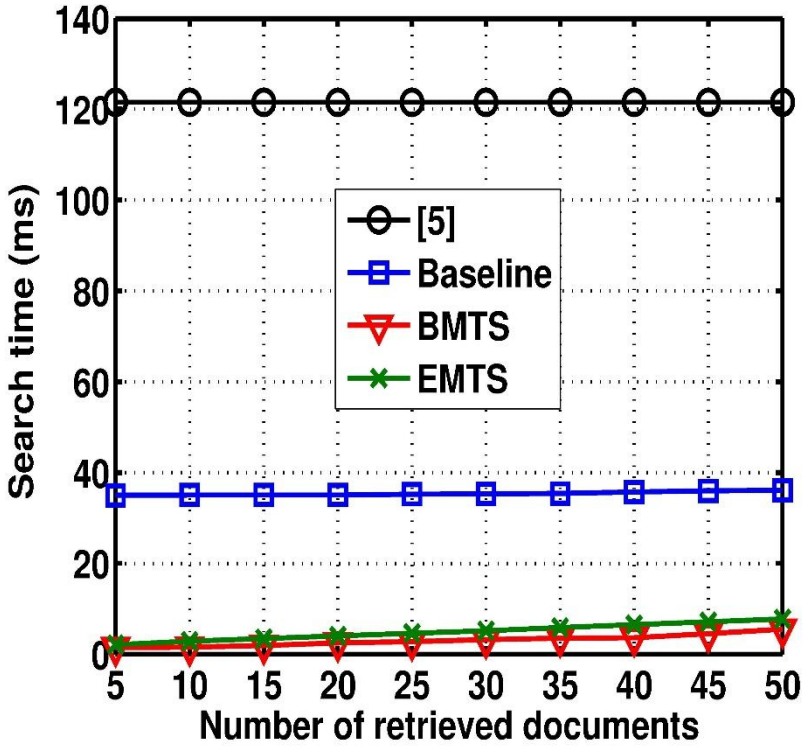
- Searching the best 3 objects



Performance analysis

- Search efficiency





Extending to Fuzzy Search

- Single keyword fuzzy search
- Multiple keyword fuzzy search

Thank You !

Acknowledgement of the Funding Agency: NSF