Security for Cloud & Big Data

CS 161: Computer Security

Prof. David Wagner

April 25, 2016

Awesome Project 2 Solutions

- Honorable mention:
  Vincent Wang and **John Choi**

- Honorable mention:
  Emily Scharff and **Sherdil Niyaz**

- Grand prize:
  Roger Chen

Awesome Project 2 Solutions

- Honorable mention:
  Vincent Wang and **John Choi** – super-efficient updates (6-9x better than our target!) using a log of changes, in just 300 lines of code
- Honorable mention:
  Emily Scharff and **Sherdil Niyaz** – elegant scheme for revocation: Alice creates a separate “telescope” (symmetric key) for each user she shares with, and keeps track of them
- Grand prize:
  Roger Chen – beautiful log-based scheme, coalesces updates in download(); only submission to pass *all* tests!

Big Data in the Cloud

Trends in computing:

- “Big data”: Easy to collect lots and lots of data about us
- “Cloud computing”: Cheaper to store data in the cloud, and do computation there

What are the security and privacy implications of these trends?

Big Data in the Cloud

Trends in computing:

- "Big data": Easy to collect lots and lots of data about us
- "Cloud computing": Cheaper to store data in the cloud, and do computation there

What are the security and privacy implications of these trends?

- Privacy – companies know a lot about us
- Data security – a security breach exposes all our data

Potential Solutions

Some possible ways to mitigate the threat:

- Policy: Minimize data collection or retention, limit who can access stored data or for what purposes
- Technology: Encrypt data while it is stored on cloud servers

Potential Solutions

Some possible ways to mitigate the threat:

- Policy: Minimize data collection or retention, limit who can access stored data or for what purposes
- Technology: Encrypt data while it is stored on cloud servers – *but then how can they do any useful computation on our data?*

Example: Project 2 + Search

- My document is stored in the cloud on a server, encrypted, as per Project 2, so I don't have to trust the server.
- But I also want to be able to do keyword search over all my documents to look for matches, without having to download and decrypt all my documents.

Example: Project 2 + Search

- My document is stored in the cloud on a server, encrypted, as per Project 2, so I don't have to trust the server.
- But I also want to be able to do keyword search over all my documents to look for matches, without having to download and decrypt all my documents.
- *How can I search in encrypted documents?*

Solution #1: Deterministic Enc.

- One solution: Each word *w* is encrypted separately and deterministically:

$$\text{DetEnc}_k(w) = \text{AES-CBC}_k(w) \text{ with IV} = \text{SHA256}(w)$$

- Advantage: Keyword searches just work, as long as I encrypt the keyword I'm searching on.

- Security?

Solution #1: Deterministic Enc.

- One solution: Each word *w* is encrypted separately and deterministically:

$$\mathrm{DetEnc}_k(w) = \text{AES-CBC}_k(w) \text{ with IV} = \mathrm{SHA256}(w)$$

- Advantage: Keyword searches just work, as long as I encrypt the keyword I'm searching on.
- Security? This leaks a lot of data about my docs.

Solution #2: Verifiable Enc.

- For each word w, store

  r, SHA256(r || DetEnc$_k$(w))

  where r is random and different each time, and DetEnc$_k$(w) is deterministic encryption as before.

- To search for word w, send x = DetEnc$_k$(w) to server. For each r, y on the server, server can test whether SHA256(r || x) = y.

- Security?

Solution #2: Verifiable Enc.

- For each word w, store

  r, SHA256(r || DetEnc$_k$(w))

  where r is random and different each time, and DetEnc$_k$(w) is deterministic encryption as before.

- To search for word w, send x = DetEnc$_k$(w) to server. For each r, y on the server, server can test whether SHA256(r || x) = y.

- Security? Leaks data about the keywords I search for, but not other words.

Solution #3: Encrypted Indices

- Standard search index: a dict that maps word *w* to list of names of documents that contain *w*.

  { 'giraffe': [1, 3, 17], 'egotistical': [5, 17, 20], ... }

- Encrypted index: encrypt each entry separately.

  { H(*k*, 'giraffe'): E_k([1,3,17]),
  H(*k*, 'egotistical'): E_k([5,17,20]) }

- To search for 'giraffe', send *x* = H(*k*, 'giraffe') to server, get back encrypted list, and decrypt it.

Security overview

- Talk to a partner, fill in the following chart:

Scheme	Time for one query	Secure for common words?	Secure for rare words?
Deterministic encrypt	O(1)		
Verifiable encryption	O(n)		
Encrypted index			

Security overview

- Talk to a partner, fill in the following chart:

Scheme	Time for one query	Secure for common words?	Secure for rare words?
Deterministic encrypt	O(1)	✗	✔
Verifiable encryption	O(n)	✔ (except searched)	✔
Encrypted index	O(1)	✔	✔

Case Study: Encrypted Email

- My email is stored in the cloud on a server.
- For security reasons, I want it to be stored in encrypted form, so I don’t have to trust the server.
- But I also want to be able to do keyword search on all my email.

Case Study: Encrypted Email

- My email is stored in the cloud on a server.
- For security reasons, I want it to be stored in encrypted form, so I don't have to trust the server.
- But I also want to be able to do keyword search on all my email.
- *How can I search on encrypted email?*

Case Study: Encrypted Email

- My email is stored in the cloud on a server.
- For security reasons, I want it to be stored in encrypted form, so I don’t have to trust the server.
- But I also want to be able to do keyword search on all my email.
- *How can I search on encrypted email?*
- Answer: Any of the above techniques.
(But can’t do regexp/wildcard searches, e.g., searching for “giraf*”.)

Solution for Encrypted Email

- One solution: Each word *w* is encrypted separately and deterministically:
  $E_k(w) = \text{AES-CBC}_k(w)$ where $IV = \text{SHA256}(w)$
- Advantage: Keyword searches just work, as long as I encrypt the keyword I'm searching on.
  Problem: This leaks a lot of data about my email.

Solution for Encrypted Email

- One solution: Each word w is encrypted separately and deterministically:
  $E_k(w) = \text{AES-CBC}_k(w)$ where IV = SHA256(w)
- Advantage: Keyword searches just work, as long as I encrypt the keyword I'm searching on.
  Problem: This leaks a lot of data about my email.
- More secure solution: For each word w, store
  r, SHA256(r, $E_k(w)$)
  where r is random and different each time, and $E_k(w)$ is deterministic encryption as above.
- To search for word w, send $x = E_k(w)$ to server.
  For each r, y on the server, server can test whether SHA256(r, x)=y.

Case Study: CryptDB

- Databases often get hacked. CryptDB encrypts all data in database, so you don't have to trust your database (as much).
- *How can I do SQL queries on encrypted database?*

Solution: Crypto

- Some queries can be handled with above techniques.   E.g.,
  SELECT * WHERE name=‘David’   →
  SELECT * WHERE name=0xF6C..18

- Can handle SELECT with equality match; JOIN.
  For SUM, use homomorphic crypto (next).

Homomorphic encryption

- RSA encryption is homomorphic:

  $E(a \times b) = a^3 \times b^3 = E(a) \times E(b) \pmod{n}$

  This lets you compute products of encrypted data.

- For sums, Paillier encryption (not taught in this class) has a similar homomorphic property:

  $E(a+b) = \ldots = E(a) \boxplus E(b)$

Solution: Crypto

- Some queries can be handled with above techniques. E.g.,
  SELECT * WHERE name='David' →
  SELECT * WHERE name=0xF6C..18

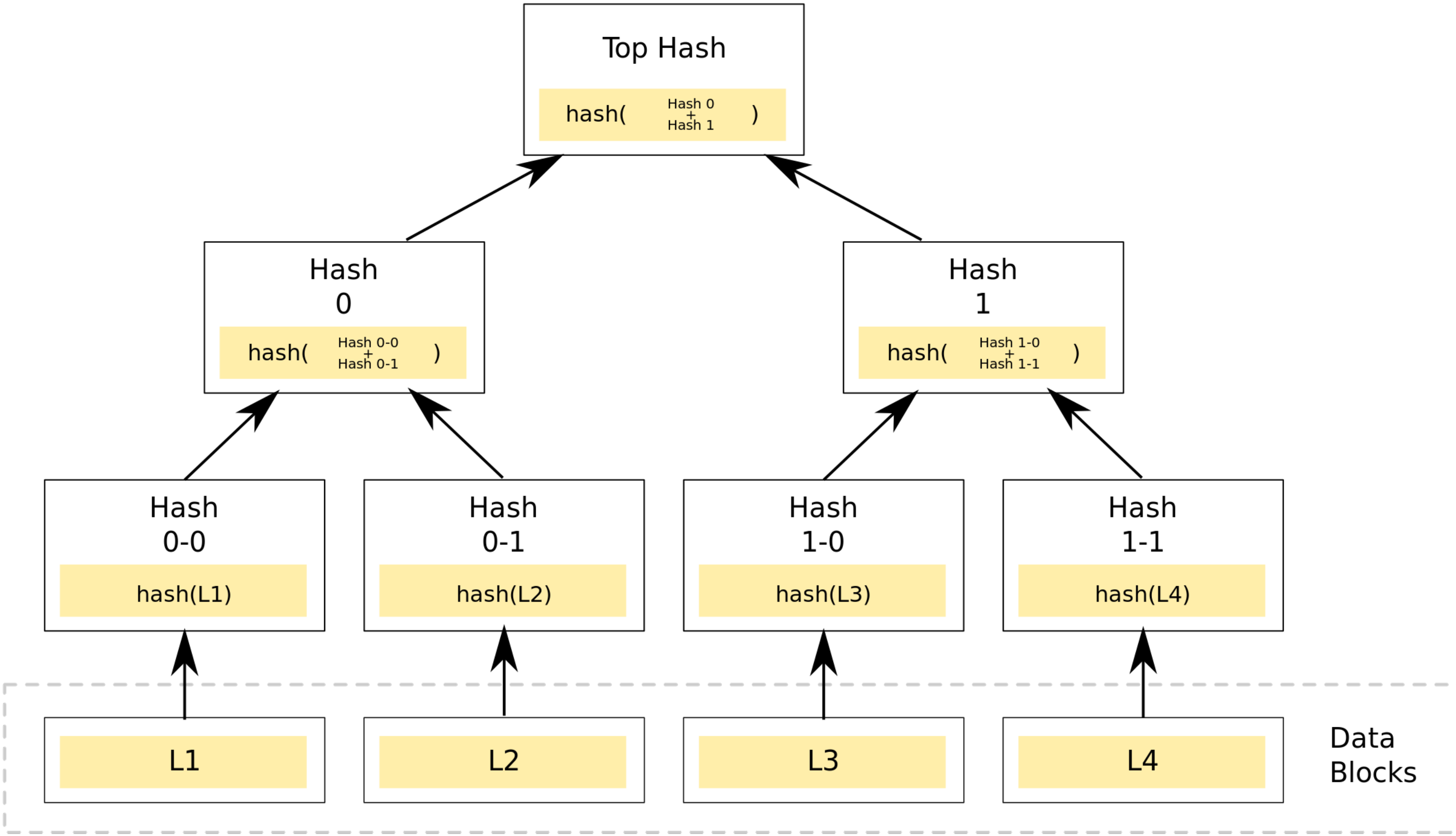
- Can handle SELECT with equality match; JOIN. For SUM, use homomorphic crypto (next).
- For all other SQL operations, download data to client and decrypt in client.

- Works surprisingly well: ~ 15% performance overhead, almost all sensitive data can be encrypted.

Integrity

- That provides confidentiality; what about integrity?
- Want to verify that any records returned by server are actually part of database (and isn't spoofed).

Merkle Tree



Top Hash
hash(Hash 0 + Hash 1)
Hash 0
hash(Hash 0-0 + Hash 0-1)
Hash 1
hash(Hash 1-0 + Hash 1-1)
Hash 0-0
hash(L1)
Hash 0-1
hash(L2)
Hash 1-0
hash(L3)
Hash 1-1
hash(L4)
L1
L2
L3
L4
Data Blocks

Takeaways

- Crypto provides a powerful way to protect data in the cloud – and allows servers to do *some* useful work on your data, without seeing the data.