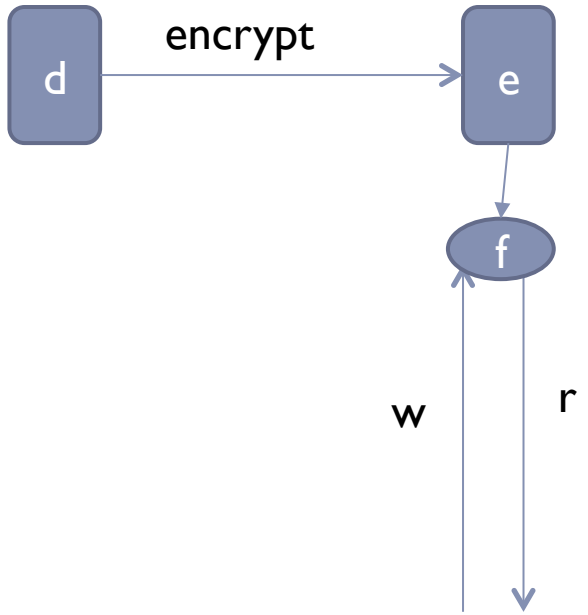
Searchable Encryption in a nutshell

Sebastian Stange, Hasso-Plattner-Institute Potsdam
04.07.2013

My first (naïve) expectations

- have a document d
- encrypt it to get e
- have a search term w
- have a function f: e × w → r

  with r being the position of w in d



- had the same expectations?
- I have to disappoint you…

BUT

still cool field of research!

Motivation

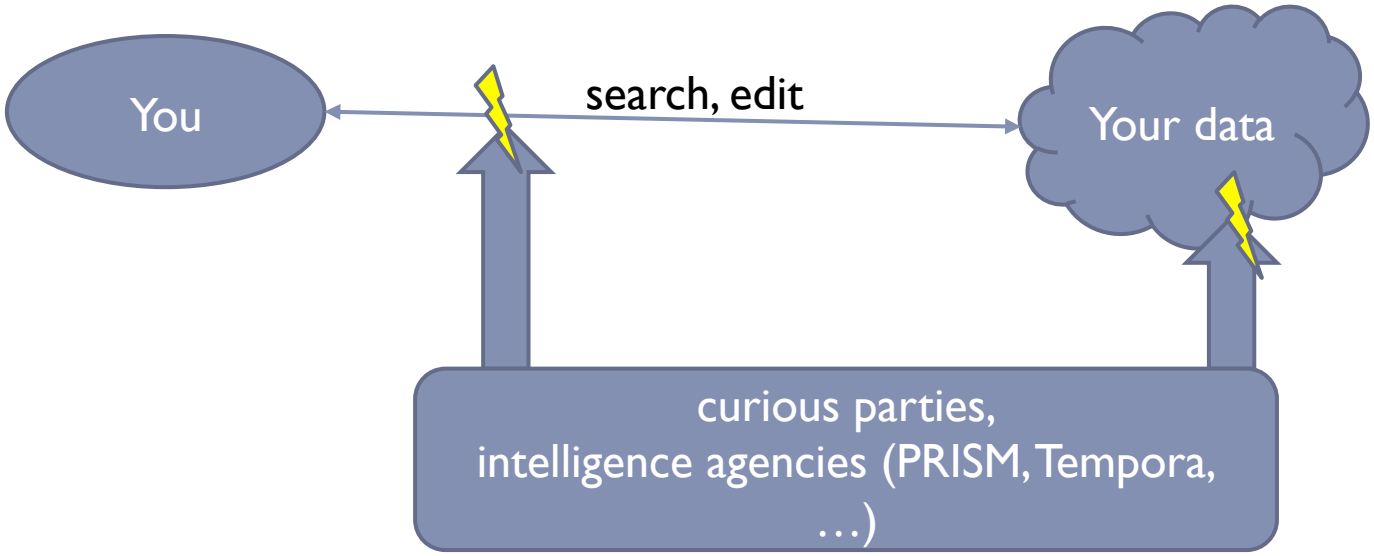
- store sensitive data on 3rd party servers (cloud)

  AND

- search & edit (update, delete)

Motivation

- assumptions:
  - you do not trust server („honest-but-curious“)
  - you do not trust the line



What is a search?

- definition of „search“:
  - ask whether a document d contains search term w
  - extended to document collections:
    - ask for all documents that contain w

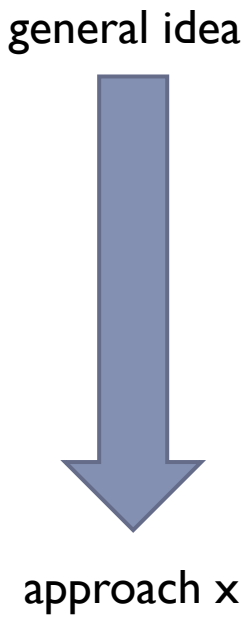
What is a search?

- search is performed using indexes
  - performance!
  - arbitrary encryption for documents!

Focus & Structure

- will try to narrow it down from a general perspective to the very concrete approaches

WHY P R O V E



Naïve approaches

- encrypt all and upload to server
- search query = download all,
  decrypt locally,
  search (& edit)

Naïve approaches

- single-user scenario:
  - assign unique id to each document
  - create index locally
    - assign id of each document to each unique word it contains
  - encrypt all documents, upload to server
  - query index locally
  - request server to transfer encrypted documents with matching ids from local index
  - ➔ no security needed

general expectations

- reasonably fast:
  - low communication overhead
  - low computation overhead (client & server)
  - low memory overhead (client & server)

  - HIGH security
    - privacy

adapted system expectations

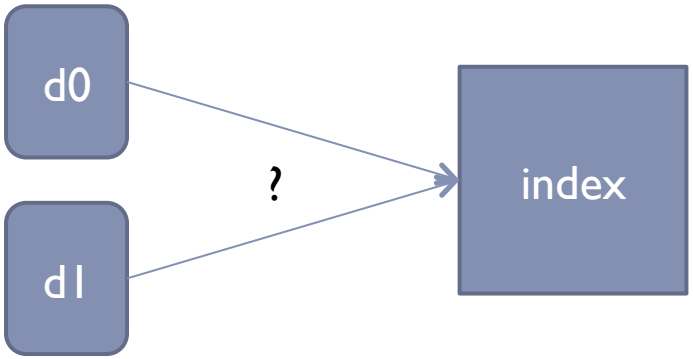
- server stores encrypted documents
- query server with encrypted search term e, generated from plain term w
- server answers with all encrypted documents containing w in their decrypted representation

detailed security expectations

- server cannot infer w from e
- server cannot infer any d from set of queried e
- server cannot infer any d from index
- access & search patterns not discoverable (privacy)

formal security requirements

- index indistinguishability, resist adaptive chosen keyword attacks (IND2-CKA)



Approaches

- (Oblivious RAMs)
- Symmetric Searchable Encryption (SSE)
- Asymmetric (efficient) Searchable Encryption (ASE or ESE)

SSE in general

- collection of 4 PT-algorithms:
  - Keygen: $s \rightarrow K_{priv}$,
  - BuildIndex: $D_{id} \times K_{priv} \rightarrow I_{D_{id}}$,
  - Trapdoor: $K_{priv} \times w \rightarrow T_w$,
  - SearchIndex: $T_w \times I_{D_{id}} \rightarrow \{0,1\}$

1st SSE approach by Goh 2003/2004

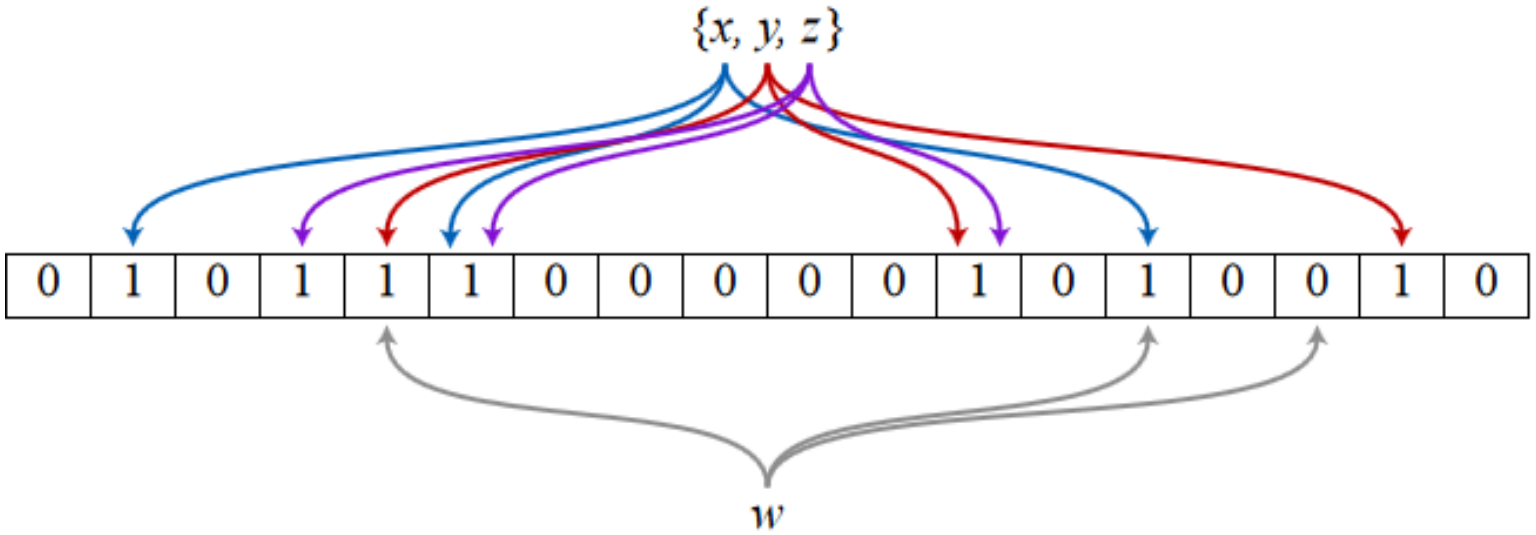
- Z-IDX
- uses Bloom filters as indexes
- pseudo random functions as trapdoors

Bloom filters - Recap

- "A Bloom filter, conceived by Burton Howard Bloom in 1970,[1] is **a space-efficient probabilistic data structure** that is used **to test whether an element is a member of a set**. False positive retrieval results are possible, but false negatives are not; i.e. a query returns either **"inside set (may be wrong)"** or **"definitely not in set"**. Elements can be added to the set, but not removed (though this can be addressed with a counting filter). The more elements that are added to the set, the larger the probability of false positives."

https://en.wikipedia.org/wiki/Bloom_filter, 01.07.2013

Bloom filters - Recap



m = 18 (size of array),
k = 3 different hash functions

https://en.wikipedia.org/wiki/Bloom_filter, 01.07.2013

1st approach – index construction

- for each unique word calculate hash using trapdoor function (HMAC-SHA1)
- with each hash calculate „codeword“ using trapdoor, which depends on document the word appears in
- insert codeword into Bloom filter for particular document
- blind the index with

  max(#wordsPerDoc) * #hashFunctions

  randomly distributed 1‘s

1st approach - search

- receive trapdoor for search term
- compute codeword for search term with each document
- test if Bloom filter for document contains codeword
  - yes/no ➔ search completed

1st approach - compelxity

- computational:
  - index generation per document: O(#wordsPerDoc)
    - all indexes: O(#documents * #avgWordsPerDoc)
  - search: O(#documents)
- space:
  - indexes: O(#uniqueWords * #documents)
    - exact sizes depend on parameters for bloom filter:

perDocumentIndexSize = #uniqueWords * c * (-$\log_2$ (#falsePositiveRate)) / ln (2)

2nd SSE approach by Ostrovsky et al. 2006

- based on a look-up table (i.e. hashmap or dictionary)
- also PRFs as trapdoors

2nd approach – index construction

Keygen(1^k): **Generate random key** $s \xleftarrow{R} \{0,1\}^k$ **and output** $K = s$.

BuildIndex$(K, \mathcal{D})$:

1. **Initialization:**
   - **scan** $\mathcal{D}$ **and build** Δ'**, the set of distinct words in** $\mathcal{D}$**. For each word** $w \in \Delta'$**, build** $\mathcal{D}(w)$**.**
2. **Build look-up table** T**:**
   a) **for each** $w_i \in \Delta'$**:**
      - **for** $1 \leq j \leq |\mathcal{D}(w_i)|$**:**
        - $\mathtt{value} = \mathsf{id}(D_{i,j})$**, where** $\mathsf{id}(D_{i,j})$ **is the** j^{th} **identifier in** $\mathcal{D}(w_i)$**;**
        - **set** $\mathtt{T}[\pi_s(w_i||j)] = \mathtt{value}$.

   b) **let** $m' = \sum_{w_i \in \Delta'} |\mathcal{D}(w_i)|$**. If** $m' < m$**, then set values for the remaining** $(m - m')$ **entries such that for all** $D \in \mathcal{D}$**, it holds that** $\mathtt{value} = \mathsf{id}(D)$ **for exactly** max **entries. Also, set the** address **field of these remaining entries to random values.**
3. **Output** $\mathcal{I} = \mathtt{T}$.

Trapdoor(w): **Output** $T_w = (T_{w_1}, \ldots, T_{w_{\max}}) = (\pi_s(w||1), \ldots, \pi_s(w||\max))$.

whiteboard

2nd approach - search

- fairly simple:

- receive trapdoors for whole word family
- retrieve document ids from look-up table using trapdoors as keys (for whole word family)

2nd approach - complexity

- computation:
  - index generation:
    - O(#uniqueWords * max(#uniqueWordsPerDoc))
  - search:
    - O(#documentsTheSearchTermAppearsIn)
- space:
  - index:
    - O(#uniqueWords * max(#uniqueWordsPerDoc))
- communication:
  - equals max(#uniqueWordsPerDoc) (query whole word family)

ASE in general

- collection of 4 PT-algorithms:
  - Keygen: $s \rightarrow K_{priv}, K_{pub}$,
  - Send: $D_{id} \times K_{pub} \rightarrow I_{D_{id}}$,
  - Trapdoor: $K_{priv} \times w \rightarrow T_w$,
  - Retrieve: $T_w \times K_{pub} \times I_{D_{id}} \rightarrow \{0,1\}$

Known applications

- three scientists from hitachi developed SSE based on homomorphic functions
- published Jan. 11, 2013 (recently!)
- incorporated it in BLAST

  (*Basic Local Alignment Search Tool*)

  ➔ genome analysis via the cloud possible, respecting privacy of "genome owner"

http://www.hitachi.com/rd/portal/story/searchable_encryption/index.html

Known applications

- **„Mitsubishi Electric Develops Searchable Encryption Platform Software“**
  (press release in „The Wall Street Journal“)

- announced July 2, 2013 (very recently!!)

- Key Features
  - 1) Robust encryption technology for secure storage in cloud services – (simple but useful access control functionality, such as group-based document sharing)
  - 2) Large data searches in 1-3 seconds, or less
  - 3) No dedicated application required in client PCs → ActiveX plug-in

- Pending patents for the technology total three in Japan and two overseas. (!)

http://online.wsj.com/article/PR-CO-20130702-913479.html

Future work

- approximate matches
  - without fuzzy query sets

Summary

- general structure of Searchable Encryption Systems
- differences in general structures of SSE and ASE
- 2 examples for SSE (both IND2-CKA)
- general ASE structure

Sources

- [1] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access Pattern disclosure on Searchable Encryption : Ramification , Attack and Mitigation."
- [2] S. Kamara, K. Lauter, and Microsoft Research, "Cryptographic cloud storage."
- [3] S. Kamara, U. C. Berkeley, and T. Roeder, "CS2 : A Searchable Cryptographic Cloud Storage System," pp. 1–25.
- [4] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12, p. 965, 2012.
- [5] M. Bellare, A. Boldyreva, and A. O. Neill, "Efficiently-Searchable and Deterministic Asymmetric Encryption," vol. 2, pp. 1–29, 2006.
- [8] Y. Chang and M. Mitzenmacher, "Privacy Preserving Keyword Searches on Remote Encrypted Data."
- [9] D. Boneh and W. E. S. Iii, "Public Key Encryption that Allows PIR Queries," no. 0430254, 2006.
- [10] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persianoz, "Public key encryption with keyword search," Advances in Cryptology- …, pp. 1–15, 2004.
- [11] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions," Journal of Cryptology, vol. 21, no. 3, pp. 350–391, Sep. 2007.
- [12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," pp. 79–88, 2006.
- [13] E. Goh, "Secure Indexes," pp. 1–19, 2004.
- [14] B. Wang, T. Chen, and F. Jeng, "Security Improvement against Malicious Servers in dPEKS Scheme," vol. 11, pp. 6–8, 2011.
- [15] Dan Boneh, Ananth Raghunathan, Gil Segev, "Function-Private Identity-Based Encryption: Hiding the Function in Functional Encryption"