A framework for searching encrypted databases

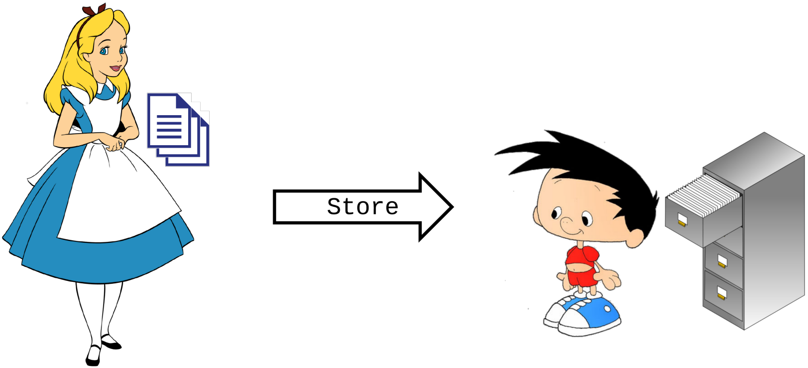
Pedro Geraldo M. R. Alves,
Diego F. Aranha

Laboratory of Security and Applied Cryptography
Instituto de Computação, Universidade Estadual de Campinas

05 de Novembro de 2016

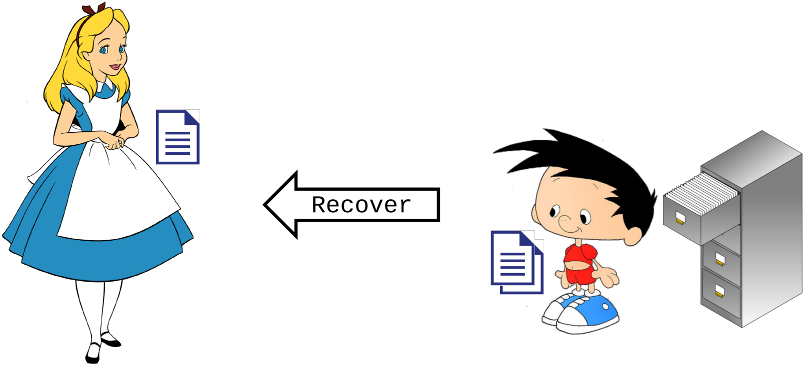
The unstrusted storage

Storing



The unstrusted storage

Recovering



Searching on encrypted datasets

Security requirements

1. Bob is not trustworthy.
   - Confidentiality must be preserved.
   - Secure storage.
2. Alice would like to occasionally retrieve subsets of documents according to predicates.
   - Communication is constrained.
   - Secure searching.

Searching on encrypted datasets

Security requirements

1. Bob is not trustworthy.
   - Confidentiality must be preserved.
   - Secure storage.
2. Alice would like to occasionally retrieve subsets of documents according to predicates.
   - Communication is constrained.
   - Secure searching.

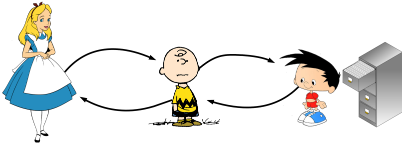
PRISM, Yahoo, Ashley Madison,...

Searching on encrypted datasets – **CryptDB**

- SQL-only.
- Open-source.

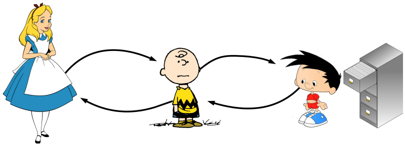
Searching on encrypted datasets – **CryptDB**

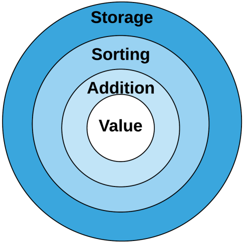
- SQL-only.
- Open-source.
- User – Aplication – Database.
  - Logged-in users are vulnerable.



Searching on encrypted datasets – **CryptDB**

- SQL-only.
- Open-source.
- User – Aplication – Database.
  - Logged-in users are vulnerable.
- Onions.
- *Selection* overhead of 6 times.





Searching on encrypted datasets – **Arx**

- Implemented on top of **MongoDB**.
- **Not** open-source (but plans to be).
- Application – Database.
- Rather than onions, data structures.
  - Requires the **previously knowledge** about what operations will be executed on each field.
- Arx-Range and Arx-Eq built over **AES** or a *deterministic* scheme.
- Arx-Eq overhead of 2 times.
- Arx-Range takes 6 *ms* in the worst case scenario for a **1M-records database** (30 times slower ?).
- Slower but more secure ?

Building blocks

Building blocks

Homomorphic encryption (HE)

Let

- **E** and **D** be a pair of **encryption and decryption** functions,
- m_1 and m_2 be plaintexts.

The pair (E, D) forms an **encryption scheme** with the **homomorphic property** for some operator $\diamond$ if and only if the following holds :

$$E(m_1) \circ E(m_2) \equiv E(m_1 \diamond m_2).$$

Building blocks

Homomorphic encryption (HE)

Let

- **E** and **D** be a pair of **encryption and decryption** functions,
- m_1 and m_2 be plaintexts.

The pair (E, D) forms an **encryption scheme** with the **homomorphic property** for some operator $\diamond$ if and only if the following holds :

$$E(m_1) \circ E(m_2) \equiv E(m_1 \diamond m_2).$$

For example, in **ElGamal's proposal**, $\circ =$ multiplication and $\diamond =$ addition.

Building blocks

Order-revealing encryption (ORE)

Let

- **E** be an **encryption** function,
- **C** be a **comparison** function,
- m_1 and m_2 be plaintexts.

The pair (E, C) is defined as an **encryption scheme** with the **order-revealing property** if and only if :

$$C(E(m_1), E(m_2)) = \begin{cases} \text{LOWER}, & \text{if } m_1 < m_2, \\ \text{EQUAL}, & \text{if } m_1 = m_2, \\ \text{GREATER}, & \text{otherwise.} \end{cases}$$

Building blocks

Order-revealing encryption (ORE)

Let

- **E** be an **encryption** function,
- **C** be a **comparison** function,
- m_1 and m_2 be plaintexts.

The pair (E, C) is defined as an **encryption scheme** with the **order-revealing property** if and only if :

$$C(E(m_1), E(m_2)) = \begin{cases} \text{LOWER}, & \text{if } m_1 < m_2, \\ \text{EQUAL}, & \text{if } m_1 = m_2, \\ \text{GREATER}, & \text{otherwise.} \end{cases}$$

For example, **[Chenette *et al.* 2015]** and **[Lewi and Wu 2016]** work.

The framework

Objective : Develop **a model** for databases capable of **storing** and **searching** on encrypted records **without any cryptographic key**.

The framework

Classes of attributes

- **Records** in a database **are composed by attributes**. These consist of a **name** and a **value** and may be **classified** according to their **purpose**.

static An *immutable* value only used for storage.

index Enables *comparison* between *index* attributes. Used for building a searchable index.

computable A mutable value. It supports the *evaluation* by a mathematical function.

The framework

Classes of attributes

- **Records** in a database **are composed by attributes**. These consist of a **name** and a **value** and may be **classified** according to their **purpose**.

static	An *immutable* value only used for storage.
index	Enables *comparison* between *index* attributes. Used for building a searchable index.
computable	A mutable value. It supports the *evaluation* by a mathematical function.

- **ORE** and **HE** schemes are natural candidates for *index* and *computable* attributes.

The framework

Building an index

In order to build a **secure and efficient** *index* we need a **Secure ORE**. This is **defined as** an ORE scheme such that $E(m) = (c_L, c_R)$ and $C\left(c_{L1}, c_{R2}\right)$.

The framework

Building an index

In order to build a **secure and efficient** *index* we need a **Secure ORE**. This is **defined as** an ORE scheme such that $E(m) = (c_L, c_R)$ and $C\left(c_{L1}, c_{R2}\right)$.

Encrypted search framework

Let **S** be a set of words, **(E,C)** the encryption and comparison functions of a **secure ORE** scheme and **(sk, pk)** secret and public keys.

$\textsc{BuildIndex}_{sk}(S)$: Output the set
$$S^* = \{c_R \ \mid (c_L, c_R) = \mathrm{E}_{sk}(w), \forall w \in S\}.$$

$\textsc{Trapdoor}_{sk}(w)$: Output the trapdoor $T_w = (c_L \mid (c_L, c_R) = \mathrm{E}_{sk}(w))$.

$\textsc{Search}_{S^*,\ r}(T_w)$: It iterates through $\mathcal{I}$ and outputs **every record such that** $C\left(T_w, w^*\right) = r$.

The framework

Database operations

- The relational algebra proposed by **[Codd 1983]** must be revisited for building a functional database.

Selection (σ) : Uses $\textsc{Search}$ to select records with the relationship $\mathbf{r} \in \{\text{LOWER}, \text{EQUAL}, \text{GREATER}\}$ when compared to **w**.

Projection (π) : In a collection of records, **selects** a subset of attributes A according to their **names**.

encrypted : Deterministic encryption or treated as *index* values.

deterministic : Selection by $(Enc_{deterministic}(a) \mid a \in A)$.

index : Selection by $\textsc{Search}$ using $(Trapdoor(a) \mid a \in A)$.

unencrypted : Standard algorithm.

The framework

Database operations

- The relational algebra proposed by **[Codd 1983]** must be revisited for building a functional database.

Selection (σ) : Uses $\textsc{Search}$ to select records with the relationship $\mathbf{r} \in \{\text{LOWER}, \text{EQUAL}, \text{GREATER}\}$ when compared to $\mathbf{w}$.

Projection (π) : In a collection of records, **selects** a subset of attributes A according to their **names**.

- encrypted : Deterministic encryption or treated as *index* values.
  - *deterministic* : Selection by $(Enc_{deterministic}(a) \mid a \in A)$.
  - *index* : Selection by $\textsc{Search}$ using $(Trapdoor(a) \mid a \in A)$.
- unencrypted : Standard algorithm.

Difference ($-$) : $A - B = \sigma_{\text{not in B}}(A)$.

Union ($\cup$) : $A \cup B = A + (B - A)$.

Intersect ($\cap$) : $A \cap B = \sigma_{\text{in B}}(A)$.

The framework

Database operations

Insert : Standard algorithm (**but records are inserted encrypted by the data owner**).

Cartesian product ($\times$) : Standard algorithm.

Update : Standard algorithm (**but only for *computable* attributes**).

Rename (ρ) : Applied on $\pi_A(\sigma_r)$.

- encrypted: Deterministic encryption or treated as *index* values.
  - *deterministic*: Replaces by $(a = Enc_{deterministic}(b) \mid a \in A)$.
  - *index*: Replaces by $(a = \textsc{Trapdoor}(b) \mid a \in A)$.
- unencrypted: Standard algorithm.

The framework

Security analysis

Pros

- Operates over an encrypted database **without the cryptographic keys**. The data owner has exclusive possession of cryptographic keys.
- Preservation of privacy **while the user is not compromised**.
- The **comparison** function may have its use **limited**.

The framework

Security analysis

Pros

- Operates over an encrypted database **without the cryptographic keys**. The data owner has exclusive possession of cryptographic keys.
- Preservation of privacy **while the user is not compromised**.
- The **comparison** function may have its use **limited**.

Cons

- Unable to **hide repeated queries**.
- Each query **reveals** the **other half** of the ciphertext.

The framework

Performance analysis

Pros

- SEARCH may be implemented with **logarithmic complexity**.
- State-of-the-art ORE proposals are built over **symmetric primitives**.

Cons

- Speed overhead.
- Space overhead.
- Does **not** support selection by **regular expressions**.

Conceptual implementation

- A proof-of-concept implementation for MongoDB was developed and **is available** to the community – `github.com/pdroalves/encrypted-mongodb`.
- Wrapper for the Python's driver.
- Implements :
  - **AES** for *static*.
  - **Lewi-Wu** for *index*.
  - **Paillier** and **ElGamal** for *computable*.
- BUILDINDEX generates an **AVL tree**.
- MongoDB **is not friendly** to custom index structures and comparators, so walking through the tree depends on a **database-external operation** at Python-side.

Conceptual implementation

Benchmark

TABLE – Attribute structure of elements in the synthetic dataset.

Name	Value type	Class
e-mail	string	*static*
firstname	string	*static*
surname	string	*static*
country	string	*static*
age	integer	*index*
text	string	*static*

Conceptual implementation

Benchmark

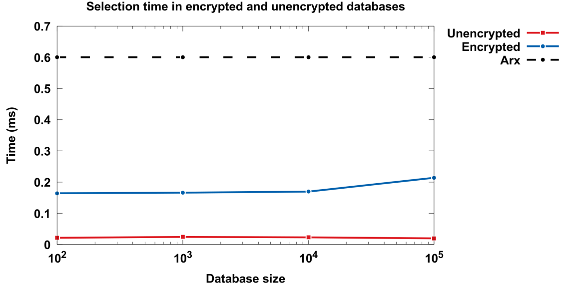


FIGURE – Time required to perform **a selection query in the worst case scenario** for an **AVL tree-based index** and 128 bits security level. The measures are the average of 100 independent executions [1].

- Space consumption increased 36%.
- Speed overhead **overhead** goes **from 7 to 11 times**.

1. Machine : Intel Xeon E5-2630 CPU at 2.60GHz, 32GB of RAM and a Seagate Barracuda SATA3-HD at 7200rpm.

Conclusion and future work

Conclusion

- We propose a framework for building **functional-encrypted databases**.
- Codd's **relational algebra was revisited** for encrypted-databases and it keeps former computational complexity.
- Privacy is **preserved** even if the database or application gets compromised.
- A proof-of-concept **implementation for MongoDB** was presented.

Future work

- Pursue a **more efficient** implementation.
- Apply the framework in a **real-world** application.

Conclusion and future work

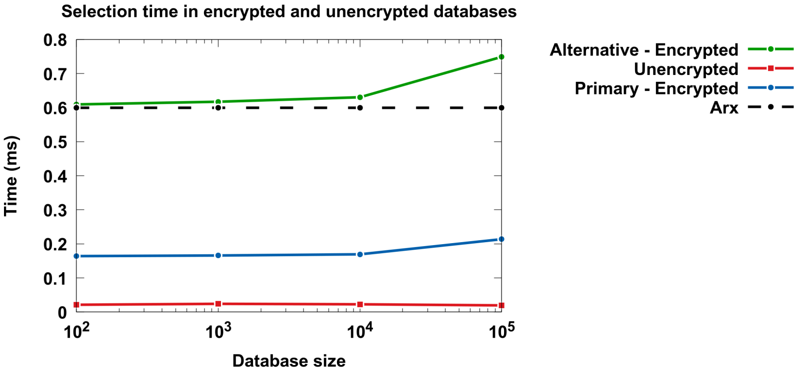


FIGURE – **Time required** to perform **a selection query in the worst case scenario** for an **AVL tree index**. Two approaches for encrypted databases are presented. The measures are the average of 100 independent executions.

A framework for searching encrypted databases

Pedro Geraldo M. R. Alves,
Diego F. Aranha

Laboratory of Security and Applied Cryptography
Instituto de Computação, Universidade Estadual de Campinas

05 de Novembro de 2016

Bibliography

- **Bosch, Hartel, Jonker, and Peter**, 2014. *A Survey of Provably Secure Searchable Encryption*.
- **Chenette** *et al.*, 2015. *Practical Order-Revealing Encryption with Limited Leakage*.
- **Lewi and Wu**, 2016. *Order-Revealing Encryption : New Constructions, Applications, and Lower Bounds*.
- **Codd**, 1983. *A relational model of data for large shared data banks*.
- **Alves**, 2016. *A proof-of-concept searchable encryption backend for MongoDB*. `github.com/pdroalves/encrypted-mongodb`.