PROPERTY-PRESERVING ENCRYPTION

GRAD SEC

NOV 07 2017

TODAY'S PAPERS

CryptDB: Protecting Confidentiality with Encrypted Query Processing

Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan
MIT CSAIL

ABSTRACT

Online applications are vulnerable to theft of sensitive information because adversaries can exploit software bugs to gain access to private data, and because curious or malicious administrators may capture and leak data. CryptDB is a system that provides practical and provable confidentiality in the face of these attacks for applications backed by SQL databases. It works by executing *SQL queries over encrypted data* using a collection of efficient SQL-aware encryption schemes. CryptDB can also *chain encryption keys to user passwords*, so that a data item can be decrypted only by using the password of one of the users with access to that data. As a result, a database administrator never gets access to decrypted data, and even if all servers are compromised, an adversary cannot decrypt the data of any user who is not logged in. An analysis of a trace of 126 million SQL queries from a production MySQL server shows that CryptDB can support operations over encrypted data for 99.5% of the 128,840 columns seen in the trace. Our evaluation shows that CryptDB has low overhead, reducing throughput by 14.5% for phpBB, a web forum application, and by 26% for queries from TPC-C, compared to unmodified MySQL. Chaining encryption keys to user passwords requires 11–13 unique schema annotations to secure more than 20 sensitive fields and 2–7 lines of source code changes for three multi-user web applications.

Categories and Subject Descriptors: H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection.*

General Terms: Security, design.

1 INTRODUCTION

Theft of private information is a significant problem, particularly for online applications [40]. An adversary can exploit software vulnerabilities to gain unauthorized access to servers [32]; curious or malicious administrators at a hosting or application provider can snoop on private data [6]; and attackers with physical access to servers can access all data on disk and in memory [23].

One approach to reduce the damage caused by server compromises is to encrypt sensitive data, as in SUNDR [28], SPORC [16], and Depot [30], and run all computations (application logic) on clients. Unfortunately, several important applications do not lend themselves to this approach, including database-backed web sites that process queries to generate data for the user, and applications that compute over large amounts of data. Even when this approach is feasible, converting an existing server-side application to this form can be difficult. Another approach would be to consider theoretical solutions such as fully homomorphic encryption [15], which allows servers to compute arbitrary functions over encrypted data, while only clients see decrypted data. However, fully homomorphic encryption schemes are still prohibitively expensive by orders of magnitude [10, 21].

This paper presents CryptDB, a system that explores an intermediate design point to provide confidentiality for applications that use database management systems (DBMSes). CryptDB leverages the typical structure of database-backed applications, consisting of a DBMS server and a separate application server, as shown in Figure 1: the latter runs the application code and issues DBMS queries on behalf of one or more users. CryptDB's approach is to execute queries *over encrypted data*, and the key insight that makes it practical is that SQL uses a well-defined set of operators, each of which we are able to support efficiently over encrypted data.

CryptDB addresses two threats. The first threat is a curious database administrator (DBA) who tries to learn private data (e.g., health records, financial statements, personal information) by snooping on the DBMS server; here, CryptDB prevents the DBA from learning private data. The second threat is an adversary that gains complete control of application and DBMS servers. In this case, CryptDB cannot provide any guarantees for users that are logged into the application during an attack, but can still ensure the confidentiality of logged-out users' data.

There are two challenges in combating these threats. The first lies in the tension between minimizing the amount of confidential information revealed to the DBMS server and the ability to efficiently execute a variety of queries. Current approaches for computing over encrypted data are either too slow or do not provide adequate confidentiality, as we discuss in §9. On the other hand, encrypting data with a strong and efficient cryptosystem, such as AES, would prevent the DBMS server from executing many SQL queries, such as queries that ask for the number of employees in the "sales" department or for the names of employees whose salary is greater than $60,000. In this case, the only practical solution would be to give the DBMS server access to the decryption key, but that would allow an adversary to also gain access to all data.

The second challenge is to minimize the amount of data leaked when an adversary compromises the application server in addition to the DBMS server. Since arbitrary computation on encrypted data is not practical, the application must be able to access decrypted data. The difficulty is thus to ensure that a compromised application can obtain only a limited amount of decrypted data. A naïve solution of assigning each user a different database encryption key for their data does not work for applications with shared data, such as bulletin boards and conference review sites.

CryptDB addresses these challenges using three key ideas:

- The first is to execute SQL queries over encrypted data. CryptDB implements this idea using a *SQL-aware encryption strategy*, which leverages the fact that all SQL queries are made up of a

SOSP '11, October 23–26, 2011, Cascais, Portugal.

Inference Attacks on Property-Preserving Encrypted Databases

Muhammad Naveed
UIUC*
naveed2@illinois.edu

Seny Kamara
Microsoft Research
senyk@microsoft.com

Charles V. Wright
Portland State University
cvwright@cs.pdx.edu

ABSTRACT

Many encrypted database (EDB) systems have been proposed in the last few years as cloud computing has grown in popularity and data breaches have increased. The state-of-the-art EDB systems for relational databases can handle SQL queries over encrypted data and are competitive with commercial database systems. These systems, most of which are based on the design of CryptDB [*SOSP 2011*], achieve these properties by making use of property-preserving encryption schemes such as deterministic (DTE) and order-preserving encryption (OPE).

In this paper, we study the concrete security provided by such systems. We present a series of attacks that recover the plaintext from DTE- and OPE-encrypted database columns using only the encrypted column and publicly-available auxiliary information. We consider well-known attacks, including frequency analysis and sorting, as well as new attacks based on combinatorial optimization.

We evaluate these attacks empirically in an electronic medical records (EMR) scenario using real patient data from 200 U.S. hospitals. When the encrypted database is operating in a steady-state where enough encryption layers have been peeled to permit the application to run its queries, our experimental results show that an alarming amount of sensitive information can be recovered. In particular, our attacks correctly recovered certain OPE-encrypted attributes (e.g., age and disease severity) for more than 80% of the patient records from 96% of the hospitals; and certain DTE-encrypted attributes (e.g., sex, race, and marital status) for more than 60% of the patient records from more than 60% of the hospitals.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

*Work done in part at Microsoft Research.

CCS'12, October 16–18, 2012, Raleigh, NC, USA.

ACM 978-1-4503-1651-4/12/10 ... $15.00.
DOI: http://dx.doi.org/10.1145/2382196.2382251

General Terms

Security, Experimentation

Keywords

inference attacks; encrypted databases; property-preserving encryption; deterministic encryption; order-preserving encryption

1. INTRODUCTION

As an increasing amount of private data is being collected and stored by corporations and governments, database security has become a critical area in both research and industry. High-profile data breaches like the Anthem breach in which a database of 80 million healthcare records was compromised or the Community Health Systems breach in which 4.5 million HIPAA protected [non-medical] records were stolen have fueled interest in database encryption techniques.

While encryption could offer some protections—particularly when the database is exfiltrated from disk—it also has serious limitations. In particular, since an encrypted database cannot be queried, it has to be decrypted in memory which means the secret key and the database are vulnerable to adversaries with memory access. In cloud settings, where a customer outsources the storage and management of its database, encryption breaks any service offered by the provider.

Encrypted search. Motivated by these limitations of traditional encryption, the area of encrypted search has emerged as one of the most active and potentially impactful areas of cryptography research. Encrypted search is concerned with the design and analysis of cryptographic techniques for searching on encrypted data, including both structured and unstructured data. There are various approaches to search on encrypted data including searchable symmetric encryption (SSE) [21, 38], fully-homomorphic encryption (FHE) [24], oblivious RAMs (ORAM) [26], functional encryption [13], and property-preserving encryption (PPE) [12,13]. All these approaches achieve different trade-offs between security, query expressiveness, and efficiency.

Leakage and inference attacks. The most secure encrypted search solutions are based on FHE and ORAM but are currently too inefficient to be of practical interest. Therefore, all known practical solutions leak some information. This leakage comes in two forms: setup leakage, which is revealed by the encrypted database (EDB) itself, and query

CRYPTDB BUILDING BLOCKS

RND *AES+CBC+random IV*

DET *AES+CBC+fixed IV*

OPE $x < y \Longrightarrow OPE_K(x) < OPE_K(y)$

HOM $HOM_K(x) * HOM_K(y) = HOM_K(x+y)$

Fully homomorphic: $F(E_K(x)) = E_K(F(x))$

SEARCH …

ORDER PRESERVING ENCRYPTION

$\mathcal{E}nc_K^{\mathsf{HG}}(\mathcal{D}, \mathcal{R}, m)$

01 $M \leftarrow |\mathcal{D}|$; $N \leftarrow |\mathcal{R}|$
02 $d \leftarrow \min(\mathcal{D}) - 1$; $r \leftarrow \min(\mathcal{R}) - 1$
03 $y \leftarrow r + \lceil N/2 \rceil$
04 If $|\mathcal{D}| = 1$ then
05 $\quad cc \xleftarrow{\$} \mathsf{TapeGen}(K, 1^{\ell_{\mathcal{R}}}, (\mathcal{D}, \mathcal{R}, 1 \| m))$
06 $\quad c \xleftarrow{cc} \mathcal{R}$
07 $\quad$ Return c
08 $cc \xleftarrow{\$} \mathsf{TapeGen}(K, 1^{\ell_1}, (\mathcal{D}, \mathcal{R}, 0 \| y))$
09 $x \xleftarrow{\$} d + \mathsf{HG}(M, N, y - r; cc)$
10 If $m \leq x$ then
11 $\quad \mathcal{D} \leftarrow \{d + 1, \ldots, x\}$
12 $\quad \mathcal{R} \leftarrow \{r + 1, \ldots, y\}$
13 Else
14 $\quad \mathcal{D} \leftarrow \{x + 1, \ldots, d + M\}$
15 $\quad \mathcal{R} \leftarrow \{y + 1, \ldots, r + N\}$
16 Return $\mathcal{E}nc_K^{\mathsf{HG}}(\mathcal{D}, \mathcal{R}, m)$

$\mathcal{D}ec_K^{\mathsf{HG}}(\mathcal{D}, \mathcal{R}, c)$

17 $M \leftarrow |\mathcal{D}|$; $N \leftarrow |\mathcal{R}|$
18 $d \leftarrow \min(\mathcal{D}) - 1$; $r \leftarrow \min(\mathcal{R}) - 1$
19 $y \leftarrow r + \lceil N/2 \rceil$
20 If $|\mathcal{D}| = 1$ then $m \leftarrow \min(\mathcal{D})$
21 $\quad cc \xleftarrow{\$} \mathsf{TapeGen}(K, 1^{\ell_{\mathcal{R}}}, (\mathcal{D}, \mathcal{R}, 1 \| m))$
22 $\quad w \xleftarrow{cc} \mathcal{R}$
23 $\quad$ If $w = c$ then return m
24 $\quad$ Else return $\perp$
25 $cc \xleftarrow{\$} \mathsf{TapeGen}(K, 1^{\ell_1}, (\mathcal{D}, \mathcal{R}, 0 \| y))$
26 $x \xleftarrow{\$} d + \mathsf{HG}(M, N, y - r; cc)$
27 If $c \leq y$ then
28 $\quad \mathcal{D} \leftarrow \{d + 1, \ldots, x\}$
29 $\quad \mathcal{R} \leftarrow \{r + 1, \ldots, y\}$
30 Else
31 $\quad \mathcal{D} \leftarrow \{x + 1, \ldots, d + M\}$
32 $\quad \mathcal{R} \leftarrow \{y + 1, \ldots, r + N\}$
33 Return $\mathcal{D}ec_K^{\mathsf{HG}}(\mathcal{D}, \mathcal{R}, c)$

Figure 2: Encryption $\mathcal{E}nc^{\mathsf{HG}}$ and decryption $\mathcal{D}ec^{\mathsf{HG}}$ algorithms for our hypergeometric distribution-based OPE scheme, $\mathsf{OPE}^{\mathsf{HG}}[\mathsf{TapeGen}]$.

SEARCHABLE ENCRYPTION

Alice is tall and Alice is small

SEARCHABLE ENCRYPTION

Alice is tall and Alice is small

REMOVE REPETITIONS

Alice is tall and small

SEARCHABLE ENCRYPTION

Alice is tall and Alice is small

REMOVE REPETITIONS

Alice is tall and small

PERMUTE POSITIONS

tall small Alice and is

SEARCHABLE ENCRYPTION

Alice is tall and Alice is small

REMOVE REPETITIONS

Alice is tall and small

PERMUTE POSITIONS

tall small Alice and is

PAD AND ENCRYPT [46]

SEARCHABLE ENCRYPTION

PROBLEM

Store these (encrypted) on an untrusted server

$W_1, \ldots, W_N$

Search for W_i

SEARCHABLE ENCRYPTION

PROBLEM

Store these (encrypted) on an untrusted server

$W_1, \ldots, W_N$

Search for W_i

SCHEME 0

PRNG: generates $S_1, \ldots, S_N$

Stream cipher

Cannot guess without knowing the original seed

Store: $W_i \oplus S_i$

Lookup: Send each S_i and W? Send seed and W?

SEARCHABLE ENCRYPTION

PROBLEM

$W_1, \ldots, W_N$

Store these (encrypted) on an untrusted server

Search for W_i

SCHEME 0

Stream cipher

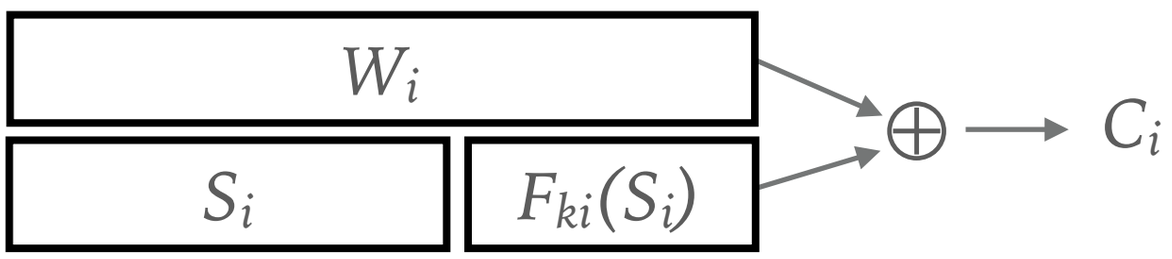
PRNG: generates $S_1, \ldots, S_N$

Cannot guess without knowing the original seed

SCHEME 1

PRF F_k

Store: $W_i \oplus S_i$ Lookup: Send each S_i and W? Send seed and W?



Store: $W_i \oplus \langle S_i, F_{ki}(S_i) \rangle$

Lookup: Send W, k_i's Server checks: $F_{ki}([C_i \oplus W]_{n-m}) = [C_i \oplus W]^m$

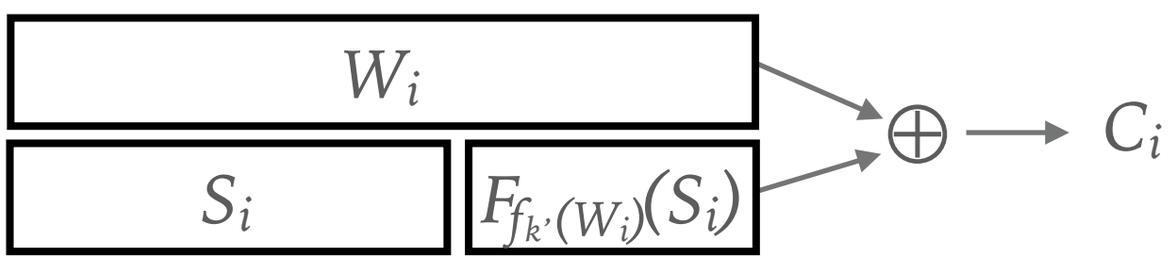
First n-m bits *Last m bits*

SEARCHABLE ENCRYPTION

SCHEME 2

Make the keys functions of the words themselves

Don't reveal keys $k_i = f_{k'}(W_i)$ never reveal k'



Store as before

Lookup: Send W, $f_{k'}(W_i)$ Server checks as before

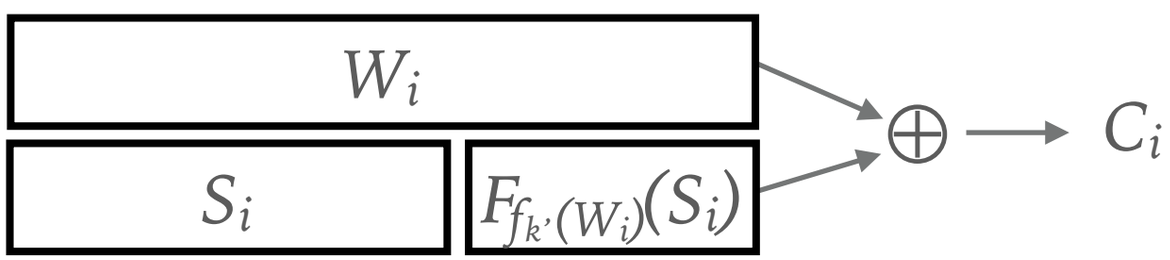
SEARCHABLE ENCRYPTION

SCHEME 2

Don't reveal keys

Make the keys functions of the words themselves

$k_i = f_{k'}(W_i)$ never reveal k'



Store as before

Lookup: Send W, $f_{k'}(W_i)$ Server checks as before

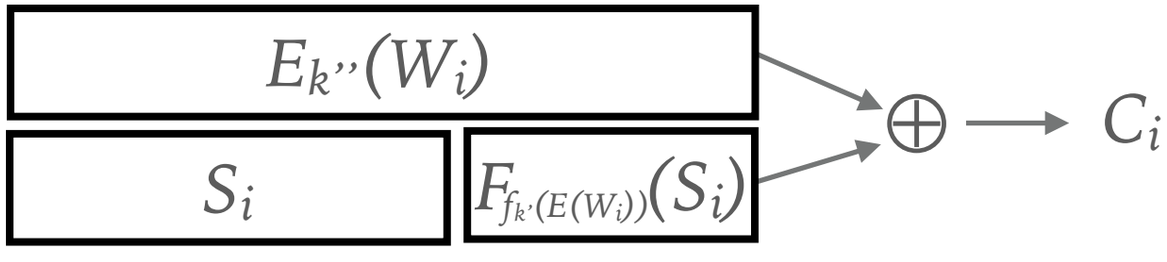
SCHEME 3

Don't reveal word

Basic idea: encrypt the word first ($E_{k''}(W_i)$ instead of W_i)

Problem 1: Randomized encryption would require sending all IVs

$\Longrightarrow$Use deterministic encryption



SEARCHABLE ENCRYPTION

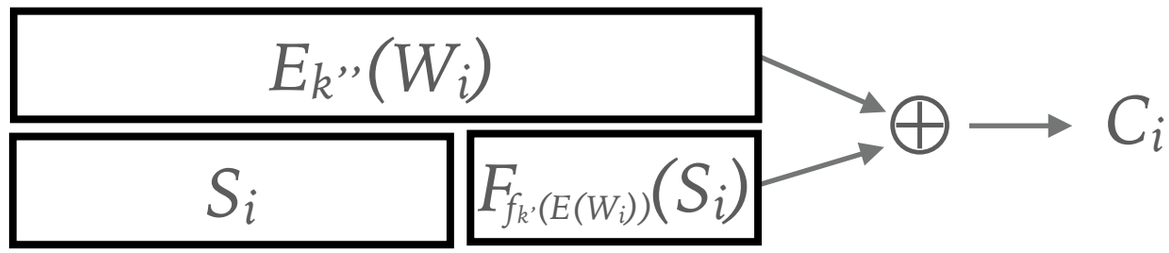
SCHEME 3

Basic idea: encrypt the word first ($E_{k''}(W_i)$ instead of W_i)

Don't reveal word

Problem 1: Randomized encryption would require sending all IVs

$\Longrightarrow$*Use deterministic encryption*



Problem 2: How do you decrypt? Need the last m bits of $E_{k''}(W_i)$

SEARCHABLE ENCRYPTION

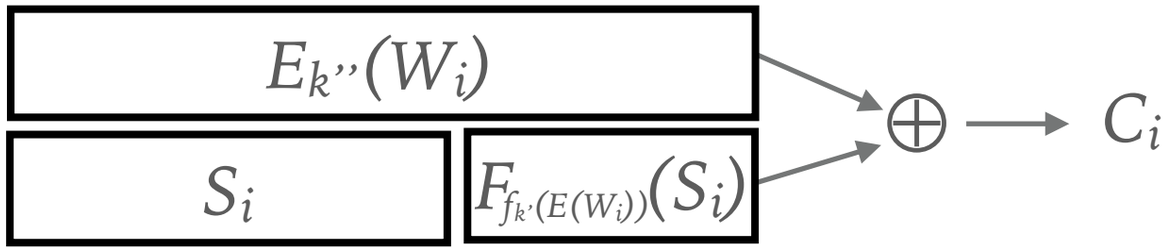
SCHEME 3

Basic idea: encrypt the word first ($E_{k''}(W_i)$ instead of W_i)

Don't reveal word

Problem 1: Randomized encryption would require sending all IVs

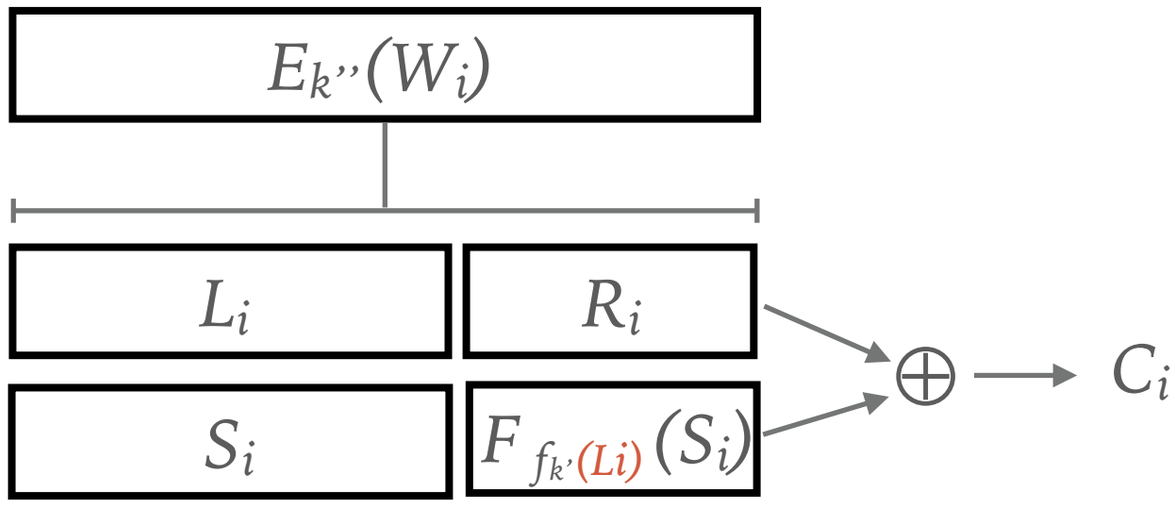
$\Longrightarrow$Use deterministic encryption



Problem 2: How do you decrypt? Need the last m bits of $E_{k''}(W_i)$

SCHEME 4

Split the ciphertext



Lookup: Send $E_{k''}(W)$, $f_{k'}(L)$

Server checks as before

CRYPTDB BUILDING BLOCKS

RND $AES+CBC+random\ IV$

DET $AES+CBC+fixed\ IV$

OPE $x < y \Longrightarrow OPE_K(x) < OPE_K(y)$

HOM $HOM_K(x) * HOM_K(y) = HOM_K(x+y)$

$Fully\ homomorphic: F(E_K(x)) = E_K(F(x))$

SEARCH $basic\ idea: E_k(W_i) \oplus \langle S_i, F_{Ki}(S_i) \rangle \qquad K_i = f_{k'}([E_k(W_i)]_{n-m})$

$To\ search,\ give\ K_i\ and\ E_k(W_i)$

CRYPTDB BUILDING BLOCKS

ONIONS

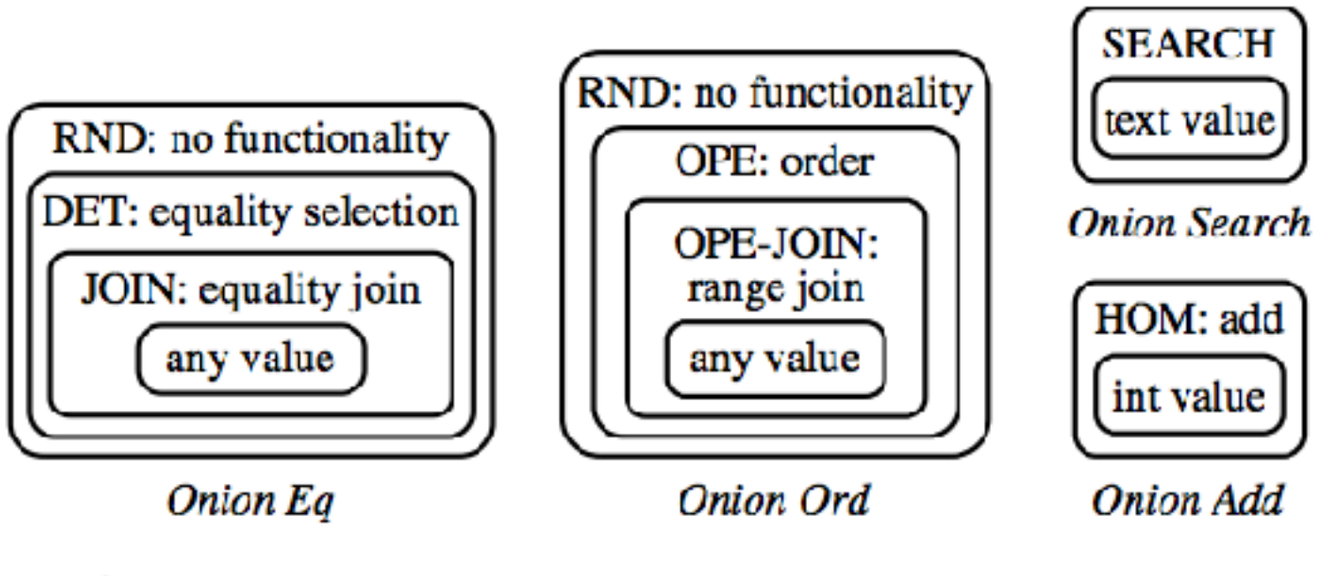


Figure 2: Onion encryption layers and the classes of computation they allow. Onion names stand for the operations they allow at some of their layers (Equality, Order, Search, and Addition). In practice, some onions or onion layers may be omitted, depending on column types or schema annotations provided by application developers (§3.5.2). DET and JOIN are often merged into a single onion layer, since JOIN is a concatenation of DET and JOIN-ADJ (§3.4). A random IV for RND (§3.1), shared by the RND layers in *Eq* and *Ord*, is also stored for each data item.

Peel off the layers as you need them

Once removed, can never un-reveal

CRYPTDB OPERATIONS

```
SELECT ID FROM Employees WHERE Name = 'Alice',
```

```
UPDATE Table1 SET
    C2-Eq = DECRYPT_RND(K_{T1,C2,Eq,RND}, C2-Eq, C2-IV),
```

```
SELECT C1-Eq, C1-IV FROM Table1 WHERE C2-Eq = x7..d,
```

Equi-joins: FROM X,Y where X.id = Y.id

Known ahead of time:
Encrypt with the same key across columns using DET

Not known ahead of time:
JOIN-ADJ

JOIN-ADJ (ADJUSTABLE JOIN)

Cryptographic hash that can be re-keyed without revealing information

$$\text{JOIN-ADJ}_K(v) := P^{K \cdot \text{PRF}_{K_0}(v)},$$

$$\Delta K = K / K'$$

$$\begin{aligned}(\text{JOIN-ADJ}_{K'}(v))^{\Delta K} &= P^{K' \cdot \text{PRF}_{K_0}(v) \cdot (K/K')} \\ &= P^{K \cdot \text{PRF}_{K_0}(v)} = \text{JOIN-ADJ}_K(v).\end{aligned}$$

ATTACKS

FREQUENCY ANALYSIS "Developed in the 9th century"

Deterministic encryption (ECB) reveals frequency

ℓ_p-OPTIMIZATION ATTACKS

Find an assignment from ciphertexts to plaintexts that minimizes a cost function

SORTING ATTACKS

Order-preserving encryption reveals .. order

CUMULATIVE ATTACK

Order-preserving encryption needs high entropy

ATTACKS ON DTE

Compare the histograms of ciphertexts to histograms of auxiliary data

- **Frequency-An**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\boldsymbol{\psi} \leftarrow \mathsf{vSort}(\mathsf{Hist}(\mathbf{c}))$; *Ciphertext*
  2. compute $\boldsymbol{\pi} \leftarrow \mathsf{vSort}(\mathsf{Hist}(\mathbf{z}))$; *Auxiliary data*
  3. output $\alpha : \mathbb{C}_k \to \mathbb{M}_k$ such that

$$\alpha(c) = \begin{cases} \boldsymbol{\pi}\left[\mathsf{Rank}_{\boldsymbol{\psi}}(c)\right] & \text{if } c \in \mathbf{c}; \\ \perp & \text{if } c \notin \mathbf{c}. \end{cases}$$

Match the rankings

- ℓ_p**-Optimization**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\boldsymbol{\psi} \leftarrow \mathsf{Hist}(\mathbf{c})$;
  2. compute $\boldsymbol{\pi} \leftarrow \mathsf{Hist}(\mathbf{z})$;
  3. output $\arg\min_{X \in \mathbb{P}_n} \|\boldsymbol{\psi} - X \cdot \boldsymbol{\pi}\|_p$; *A more general formulation*

ATTACKS ON OPE

Exploit the fact that the order is revelatory…

- **Sorting-Atk($\mathbf{c}$):**
  1. compute $\boldsymbol{\psi} \leftarrow \mathsf{vSort}(\mathsf{Unique}(\mathbf{c}))$;
  2. compute $\boldsymbol{\pi} \leftarrow \mathsf{vSort}(\mathbb{M}_k)$;
  3. output $\alpha : \mathbb{C}_k \rightarrow \mathbb{M}_k$ such that:

$$\alpha(c) = \begin{cases} \boldsymbol{\pi}[\mathsf{Rank}_{\boldsymbol{\psi}}(c)] & \text{if } c \in \mathbf{c}; \\ \perp & \text{if } c \notin \mathbf{c}. \end{cases}$$

Order, not frequency like DTE

ATTACKS ON OPE

…or that there is low entropy

- **Cumulative-Atk**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\boldsymbol{\psi} \leftarrow \mathsf{Hist}(\mathbf{c})$ and $\boldsymbol{\varphi} \leftarrow \mathsf{CDF}(\mathbf{c})$;
  2. compute $\boldsymbol{\pi} \leftarrow \mathsf{Hist}(\mathbf{z})$ and $\boldsymbol{\mu} \leftarrow \mathsf{CDF}(\mathbf{z})$;
  3. output

$$\arg\min_{X \in \mathbb{P}} \sum_{i=1}^{|\mathbb{M}_k|} \left(\left| \psi_i - X_i \cdot \boldsymbol{\pi} \right| + \left| \varphi_i - X_i \cdot \boldsymbol{\mu} \right| \right)$$

Intuitively, if a given OPE ciphertext is greater than 90% of the ciphertexts in the encrypted column c, then we should match it to a plaintext that also is greater than about 90% of the auxiliary data z.

BILINEAR MAPS

For any generator $g \in \mathbb{G},\ e(g^a, g^b) = e(g, g)^{ab}$.

Public key scheme: *Private key a* *Public key* g^a

Signature scheme: *Signature* $= H(m)^a$

Verify: $e(H(m)^a, g) = e(H(m), g^a)$

Multisignature scheme: *Signatures* $= H(m)^{a1}, \ldots, H(m)^{an}$

Multisignature $= H(m)^{a1} * \ldots * H(m)^{an}$