THE PRIVACY PUZZLE FOR IoT

We have sensors everywhere, including in very sensitive settings.

They are capturing information you definitely don’t want to share.

… seemingly arguing for brilliant sensors that do all the computing.

- But sensors are power and compute-limited.
- Sometimes, only cloud-scale datacenters can possibly do the job!
THINGS THAT CAN ONLY BE DONE ON THE CLOUD

Training models for high quality image recognition and tagging. Classifying complex images.

High quality speech, including regional accents and individual styles.

Correlating observations from video cameras with shared knowledge

- Example: A smart highway where we are comparing observations of vehicles with previously computed motion trajectories
- Is Bessie the cow likely to give birth soon? Will it be a difficult labor?
- What plant disease might be causing this form of leaf damage?
But the Cloud is Not Good on Privacy

Many cloud computing vendors are incented by advertising revenue.

- Google just wants to show ads that the user will click on.
- Amazon wants to offer products this user might buy.

Consider medications: a big business in America. But to show a relevant ad for a drug to treat mental health, or diabetes, entails knowing the user’s health status.

Even *showing* the ad could leak information that a third party, like the ISP carrying network traffic, might “steal”.
THE LAW CAN’T HELP (YET)

Lessing: “East code versus West code”.

Main points:

➤ The law is far behind the technology curve, in the United States.
➤ Europe may be better, but is a less innovative technology community.
➤ So our best hope is to just build better technologies here.
We should separate cloud providers into two groups.

One group of cloud providers has an inherent motivation to violate privacy for revenue reasons and will “fight against” constraints.

- Here we need to block their effort to spy on the computation.

A second group doesn’t earn their revenue with ads.

- These cloud vendors might cooperate to create a secure and private model.
TRAFFIC ANALYSIS ATTACKS

Some attacks don’t actually try to “see” the actual data.

Instead the attacker might just try to monitor the system carefully, as a way to see who is talking to whom, or sending big objects.

A malicious operator can use this as indirect evidence, or try and disrupt the computation at key moments to cause trouble.
These are exploits that leave a system unchanged, but find ways of extracting keys or other sensitive information during normal interactions.

For example, suppose that if I check my account at AFCU, various account information and personal information stays on their (cloud hosted) server.

But then suppose that a negative-length web page “put” operation is done and as part of the error code, the server tries to return the bad argument – but instead sends back a snapshot of its memory, revealing my private data.
OUR CLASS LOOKS AT DEPLOYING ML TO THE EDGE. EVEN AN ML MODEL CAN BE INVERTED

Machine learning systems generally operate in two stages

- Given a model, they use labeled data to “train” the model (like fitting a curve to a set of data points, by finding parameters to minimize error).
- Then the active stage takes unlabeled data and “classifies” it by using the model to estimate the most likely labels from the training set.

The model will look like a vector of numbers. At a glance it doesn’t seem to encode the information it learned.
INVERTING A MACHINE-LEARNED MODEL

But such a model *might* encode private data.

For example, a model trained on your activities in your home might “know” all sorts of very private things even if the raw input isn’t retained!

It turns out that for certain input, like data that is all 0’s or all 1’s, or that has all 0’s and a single 1, the classifier output will reveal a (noisy) version of the input it was trained on. The attacker does a legitimate operation but learns your secrets.
UNCOOPERATIVE PROVIDER

The concern arises that the provider of some service might not have done a great job of protecting the data that we send to that service.

So what can we do? If the provider is very cooperative, we could audit their code and design... but few providers are willing to allow that.

Or, we could try and run their cloud platform inside a locked box.
SOUNDS PRETTY BAD!

If our cloud provider wants to game the system, there are a million ways to evade constraints, and they may even be legal!

So realistically, with an uncooperative cloud operator, our best bet is to just not use their cloud.

Even hybrid cloud models seem to be infeasible if you need to protect sensitive user data.
Intel has created special hardware to assist for this case: iSGX. Stands for Software Guard Extensions.

Basically, they offer a way to run in a “secure context” within a vendor’s cloud. If the operator wanted to, it can’t peek into the execution context.

Let’s look closely at how iSGX works.
DEEP DIVE 1: SGX

Let’s drill down on the concrete options.

First we will look closer at SGX, since this is a product from a major vendor.
The cloud launches the SGX program, which was supplied by the client.

The program can now read data from the cloud file system or accept a secured TCP connection (HTTPS) from an external application.

The client sends data, and the SGX-secured enclave performs the task and sends back the result. The cloud vendor can only see encrypted information, and never has any access to decrypted data or code.
SGX EXAMPLE

External client system, or IoT Sensor

HTTPS connection (secure!)

Drat! I can’t see anything!

Intel.com
In itself, SGX won’t protect against monitoring attacks.

And it can’t stop someone from disrupting a connection or accosting a user and saying “why are you using this secret computing concept? Tell me or go to jail!”

And it is slow…
SGX RECEPTION HAS BEEN MIXED

Some adoption, but performance impact is a continuing worry.

There have been some successful exploits against SGX that leverage Intel’s hardware caching and prefetching policies. ("Leaks")

Using SGX requires substantial specialized expertise. And SGX can’t leverage specialized hardware accelerators, like GPU or TPU or even FPGA (they could have “back channels” that leak data).
If the vendor is willing to work with the cloud developer many new options emerge. Such a vendor guarantees: “We won’t snoop, and we will isolate users so that other users can’t snoop”.

A first simple idea is for the vendor to provide a guaranteed “scrubbing” for container virtualization.

- Containers that start in a known and “clean” runtime context.
- After the task finishes, they clean up and leave no trace at all.
ORAM MODEL

ORAM: Oblivious RAM (multiuser system that won’t leak information)

Idea here is that if the cloud operator can be trusted but “other users” on the same platform cannot, we should create containers that leak no data.

Even if an attacker manages to run on the same server, they won’t learn anything. All leaks are blocked (if the solution covered all issues, that is)

Turns out to be feasible with special design and compilation techniques
If the cloud vendor is able to “set aside” some servers, but can’t provide a private network, these tools let us create a form of VPN in which traffic for application A shares the network with traffic for other platforms, but no leakage occurs.

In practice the approach is mostly via cryptography.

For this reason, “traffic analysis” could still reveal some data.
Vendor or µ-service developer will need to implement a similar “leave no trace” guarantee.

Use cryptography to ensure that data on the wire can’t be interpreted

- With FPGA bump-in-the-wire model, this can be done at high speeds.
- So we can pass data across the cloud message bus/queue safely as long as the message tag set doesn’t reveal secrets.
- Cloud vendor could even audit the µ-services, although this is hard to do and might not be certain to detect private data leakage.
Many applications turn out to need to create a single database with data from multiple clients, because some form of “aggregated” data is key to what the µ-service is doing.

- Most customers who viewed product A want to compare with B.
- If you liked that book, you will probably like this one too.
- People like you who live in Ithaca love Gola Osteria.
- 88% of people with this gene variant are descended from Genghis Khan.
ISSUE WITH DATABASE QUERIES

Many people assume that we can anonymize databases, or limit users to queries that sum up (“aggregate”) data over big groups.

But in fact it is often surprisingly easy to de-anonymize the data, or use known information to “isolate” individuals.

- How many bottles of wine are owned by people in New York State that have taught large MEng-level cloud computing courses?
- Seems to ask about a large population, but actually asks about me!
BEST POSSIBLE? DIFFERENTIAL PRIVACY

Cynthia Dwork has invented a model called “Differential Privacy”.

We put our private database on a trusted server. It permits queries (normally, aggregation operations like average, min, max) but not retrieving individual data. **And it injects noise into results.**

Noise level can be tuned to limit the rate at which leakage occurs.
BOTTLES OF WINE QUERY

For example, if the aggregation query includes a random extra number in the range \([-10000,10000]\), then an answer like “72” tells you nothing about Ken’s wine cellar.

There are several ways to add noise, and this is a “hot topic”.

But for many purposes, noisy results aren’t very useful.

➢ “I can’t see to the right. How many cars are coming?”
Building systems that compute on encrypted data

Raluca Ada Popa
MIT PhD, now a professor at Berkeley

http://www.cs.cornell.edu/courses/cs5412/2021sp
Compromise of confidential data is prevalent.
Problem setup

no computation
storage
encryption

computation
databases, web applications, mobile applications, machine learning, etc.

http://www.cs.cornell.edu/courses/cs5412/2021sp
Current systems strategy

Prevent attackers from breaking into servers
Lots of existing work

- Checks at the operating-system level
- Language-based enforcement of a security policy
- Static or dynamic analysis of application code
- Checks at the network level
- Trusted hardware

...
Data still leaks even with these mechanisms because attackers eventually break in!
<table>
<thead>
<tr>
<th>Attacker:</th>
<th>Reason they succeed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>hackers</td>
<td>software is complex</td>
</tr>
<tr>
<td>cloud employees</td>
<td>insiders: legitimate server access!</td>
</tr>
<tr>
<td>increasingly many companies store data on external clouds</td>
<td>e.g., physical access</td>
</tr>
<tr>
<td>government</td>
<td>accessed private data according to</td>
</tr>
</tbody>
</table>

http://www.cs.cornell.edu/courses/cs5412/2021sp
[Raluca Popa’s] work

Systems that protect confidentiality even against attackers with access to all server data
My approach

Servers store, process, and compute on encrypted data in a practical way

http://www.cs.cornell.edu/courses/cs5412/2021sp
Computing on encrypted data in cryptography

[Rivest-Adleman-Dertouzos’78]

Fully homomorphic encryption (FHE) [Gentry’09]

prohibitively slow, e.g., slowdown $\times 1,000,000,000$

My work: *practical systems*

real-world performance + large class of real applications + meaningful security

http://www.cs.cornell.edu/courses/cs5412/2021sp
My contributions

System:

Databases: **CryptDB** [SOSP’11][CACM’12]
- mOPE, adjJOIN
  [Oakland’13]

Web apps: **Mylar** [NSDI’14]
- multi-key search

Mobile apps: **PrivStats** [CCS’11]
- VPriv [Usenix Security’09]

Server under attack:

- **DB server**
- **web app server**
- **mobile app server**

Theory:

In general: **Functional encryption** [STOC’13] [CRYPTO’13]

http://www.cs.cornell.edu/courses/cs5412/2021sp
Combine systems and cryptography

1. identify core operations needed

2. multiple specialized encryption schemes
   New schemes:
   - mOPE, adjJOIN for CryptDB
   - multi-key search for Mylar

3. Design and build system

strawman:
one generic scheme (FHE)
# My contributions

<table>
<thead>
<tr>
<th>System:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Databases:</strong> CryptDB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Web apps:</strong> Mylar</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Mobile apps:</strong> PrivStats VPriv</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Server under attack:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB server</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>web app server</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>mobile app server</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Theory:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional encryption</td>
</tr>
</tbody>
</table>

http://www.cs.cornell.edu/courses/cs5412/2021sp
First practical database system (DBMS) to process most SQL queries on encrypted data
Related work

- **Systems work:** [Hacigumus et al.’02][Damiani et al.’03][Ciriani et al’09]
  - no formal confidentiality guarantees
  - restricted functionality
  - client-side filtering

- **Theory work:**
  - General computation: FHE [Gentry’09]
    - very strong security: *forces slowdown - many queries must always scan and return the whole DB*
    - prohibitively slow (10^9x)
  - Specialized schemes [Amanatidis et al.’07][Song et al.’00][Boldyreva et al.’09]

http://www.cs.cornell.edu/courses/cs5412/2021sp
Setup

trusted client-side  under passive attack

Application  DB server

Use cases:
- Outsource DB to the cloud (DBaaS)
  - e.g. Encrypted BigQuery
- Local cluster: hide DB content from sys. admins.

http://www.cs.cornell.edu/courses/cs5412/2021sp
Setup

trusted client-side

under passive attack

Application

Proxy

DB server

encrypted DB

- Stores schema and master key
- No query execution

Plain query

transformed query

encrypted results

deencrypted results

computation on encrypted data $\approx$ regular computation

http://www.cs.cornell.edu/courses/cs5412/2021sp
Example

Randomized encryption (RND) - semantic

Actual values are not visible to cloud provider

SELECT * FROM emp

SELECT * FROM table1

<table>
<thead>
<tr>
<th>col1/rank</th>
<th>col2/name</th>
<th>col3/salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>x4be219</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x95c623</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x2ea887</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x17cea7</td>
</tr>
</tbody>
</table>
Example

```
<table>
<thead>
<tr>
<th>col1/rank</th>
<th>col2/name</th>
<th>col3/salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>x934bc1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x5a8c34</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x5a8c34</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x84a21c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x5a8c34</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x5a8c34</td>
</tr>
</tbody>
</table>
```

Randomized Encryption (RND) vs. Deterministic Encryption (DET)

Application

```
SELECT * FROM emp
WHERE salary = 100
```

Proxy

```
SELECT * FROM table1
WHERE col3 = x5a8c34
```

http://www.cs.cornell.edu/courses/cs5412/2021sp
Example

```
SELECT cdb_sum(col3) 
FROM table1
```

```
<table>
<thead>
<tr>
<th>col1/rank</th>
<th>col2/name</th>
<th>col3/salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>x9eab81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x638e54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x122eb4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x578b34</td>
</tr>
</tbody>
</table>
```

```
SELECT sum(salary) 
FROM emp
```

```
| 60 | 100 | 800 | 100 |
```

Deterministic encryption (DET) - "Summable" encryption (HOM)
Techniques

1. Use SQL-aware set of efficient encryption schemes *(meta technique!)*
   - Most SQL can be implemented with a few core operations

2. Adjust encryption of data based on queries

3. Query rewriting algorithm
## 1. SQL-aware encryption schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Construction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RND</td>
<td>AES in UFE</td>
<td>data moving</td>
</tr>
<tr>
<td>HOM</td>
<td>Paillier</td>
<td>addition</td>
</tr>
<tr>
<td>SEARCH</td>
<td>Song et al., ’00</td>
<td>word search</td>
</tr>
<tr>
<td>DET</td>
<td>AES in CMC</td>
<td>equality</td>
</tr>
<tr>
<td>JOIN</td>
<td>our new scheme</td>
<td>join</td>
</tr>
<tr>
<td>OPE</td>
<td>our new scheme</td>
<td>order</td>
</tr>
</tbody>
</table>

**Security**

- ≈ semantic security
- reveals only repeat pattern
- reveals only order

**SQL operations:**
- e.g., SELECT, UPDATE, DELETE, INSERT, COUNT
- e.g., SUM, +
- restricted LIKE
- e.g., =, !=, IN, GROUP BY, DISTINCT
- e.g., >, <, ORDER BY, ASC, DESC, MAX, MIN, GREATEST, LEAST

### Example

- $x < y$ $\iff$ $\text{Enc}(x) < \text{Enc}(y)$

*Song et al., '00*

[Oakland'13]
How to encrypt each data item?

Goals:
1. Support queries
2. Use most secure encryption schemes

Challenge: may not know queries ahead of time

<table>
<thead>
<tr>
<th>rank</th>
<th>'CEO'</th>
<th>'worker'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALL?</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>col1-RND</th>
<th>col1-HOM</th>
<th>col1-SEARCH</th>
<th>col1-DET</th>
<th>col1-JOIN</th>
<th>col1-OPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Leaks order!
Onion

http://www.cs.cornell.edu/courses/cs5412/2021sp
Onion of encryptions

Adjust encryption: strip off layer of the onion

http://www.cs.cornell.edu/courses/cs5412/2021sp
Onions of encryptions

1 column

3 columns

Onion Equality

Onion Order

Onion Search

Same key for all items in a column for same onion layer

http://www.cs.cornell.edu/courses/cs5412/2021sp
Onion evolution

- Start out the database with the most secure encryption scheme
- If needed, adjust onion level
  - Proxy gives decryption key to server
  - Proxy remembers onion layer for columns

Lowest onion level is never removed

http://www.cs.cornell.edu/courses/cs5412/2021sp
Example

```
SELECT * FROM emp WHERE rank = 'CEO'
```

Logical table:

<table>
<thead>
<tr>
<th>rank</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>'CEO'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'worker'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical table:

```
table 1:

<table>
<thead>
<tr>
<th>col1-OnionEq</th>
<th>col1-OnionOrder</th>
<th>col1-OnionSearch</th>
<th>col2-OnionEq</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://www.cs.cornell.edu/courses/cs5412/2021sp
Example (cont’d)

UPDATE table1 SET col1-OnionEq = Decrypt_RND(key, col1-OnionEq)

JOIN

SELECT * FROM table1 WHERE col1-OnionEq = xda5c0407

SELECT * FROM emp WHERE rank = ‘CEO’

http://www.cs.cornell.edu/courses/cs5412/2021sp
Security threshold

Data owner can specify minimum level of security

CREATE TABLE emp (...,
credit_card SENSITIVE integer, ...
)

RND, HOM, DET for unique fields

≈ semantic security

http://www.cs.cornell.edu/courses/cs5412/2021sp
Security guarantee

Columns annotated as sensitive have semantic security (or similar).

Encryption schemes exposed for each column are the most secure enabling queries.

equality repeats sum semantic no filter semantic

common in practice

Never reveals plaintext

http://www.cs.cornell.edu/courses/cs5412/2021sp
Limitations & Workarounds

Queries not supported:

- More complex operators, e.g., trigonometry
- Certain combinations of encryption schemes:
  - e.g., \(\text{salary + raise} > 100\text{K}\)

Use query splitting, query rewriting

HOM

http://www.cs.cornell.edu/courses/cs5412/2021sp
Implementation

No change to the DBMS!
Largely no change to apps!
Evaluation

1. Does it support real queries/applications?
2. What is the resulting confidentiality level?
3. What is the performance overhead?
Real queries/applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Encrypted columns</th>
<th># cols with queries not supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpBB</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>HotCRP</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>grad-apply</td>
<td>103</td>
<td>0</td>
</tr>
<tr>
<td>TPC-C</td>
<td>92</td>
<td>0</td>
</tr>
<tr>
<td>sql.mit.edu</td>
<td>128,840</td>
<td>1,094</td>
</tr>
</tbody>
</table>

SELECT 1/log(series_no+1.2) ...
...
... WHERE sin(latitude + PI()) ...

http://www.cs.cornell.edu/courses/cs5412/2021sp
## Confidentiality level

<table>
<thead>
<tr>
<th>Application</th>
<th>Encrypted columns</th>
<th>Min level: ≈semantic</th>
<th>Min level: DET/JOIN</th>
<th>Min level: OPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpBB</td>
<td>23</td>
<td>21</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HotCRP</td>
<td>22</td>
<td>18</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>grad-apply</td>
<td>103</td>
<td>95</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>TPC-C</td>
<td>92</td>
<td>65</td>
<td>19</td>
<td>8</td>
</tr>
<tr>
<td>sql.mit.edu</td>
<td>128,840</td>
<td>80,053</td>
<td>34,212</td>
<td>13,131</td>
</tr>
</tbody>
</table>

Most columns at semantic were less sensitive

Most columns at OPE were less sensitive
Performance

MySQL:

Application 1

Application 2

DB server throughput

Plain database

Latency

CryptDB:

Application 1

CryptDB Proxy

Application 2

CryptDB Proxy

Encrypted DB

Hardware: 2.4 GHz Intel Xeon E5620 – 8 cores, 12 GB RAM
TPC-C performance

Latency (per query): 0.10ms MySQL vs. 0.72ms CryptDB

Throughput loss over MySQL: 26%

No cryptography at the DB server in the steady state!

Homomorphic addition

http://www.cs.cornell.edu/courses/cs5412/2021sp
Adoption

http://css.csail.mit.edu/cryptdb/

Google
Encrypted BigQuery [http://code.google.com/p/encrypted-bigquery-client/]

“CryptDB was really eye-opening in establishing the practicality of providing a SQL-like query interface to an encrypted database”

“CryptDB was [..] directly influential on the design and implementation of Encrypted BigQuery.”

Úlfar Erlingsson, head of security research, Google

SAP
SEEED implemented on top of the SAP HANA DBMS

MIT Lincoln Laboratory
Encrypted version of the D4M Accumulo NoSQL engine

sql.mit.edu Users opted-in to run Wordpress over our CryptDB source code

http://www.cs.cornell.edu/courses/cs5412/2021sp
CONCERNS ABOUT CRYPTDB?

The main criticisms stem from the “strip a layer” step.

Once we reduce the level of protection, we’ve leaked some information and the remaining data is “less protected”. Raluca’s response: if you want to make use of operations like aggregation, you can’t easily avoid releasing some information.

Criticism response to Raluca: attacker might trick my code into doing the operation, and might do so in the future when some flaw in one of the crypto scheme is noticed. The logic wouldn’t protect itself in that case.
A “leave no trace” model could offer a practical way to leverage the cloud and yet not release private data to the public.

With a trusted vendor willing to audit operations and to “enclave” sensitive data computation, and clean up afterward, there is real hope for privacy without leaks.

SGX, costly but can be used where the vendor is not trusted.

For databases, techniques like CryptDB aren’t perfect but work well. Differential privacy is even better, but only if noise can be tolerated.