Cloud and Mobile Security Seminar

Lecture 3: Cryptography-based Solutions to Cloud Security Challenges

Reminder

1-page description of your project due today!

Please upload to Courseworks (not email!)

I will respond within a few days

Outline

- Review of cloud challenges and solutions
- SQL-enabled encryption
  - **CryptDB: Protecting Confidentiality with Encrypted Query Processing.** Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. In *Proceedings of SOSP*, 2011.

Next time:

- Trusted collaborative cloud storage
- Fully-homomorphic encryption

Quiz: Cloud Data Threats

Q: What bad actions can each of these actors inflict on a cloud user's data?

- Malicious, buggy, or inflexible cloud provider
- Malicious or buggy co-located user
- Hackers
- Legal investigators

Quiz: Cloud Data Threats

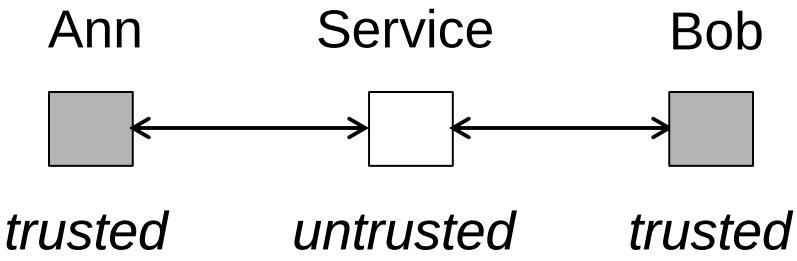
Q: What bad actions can each of these actors inflict on a cloud user's data?

- Malicious, buggy, or inflexible cloud provider
  - Sell user's data to ad or insurance companies
  - Replicate less to save on storage costs
  - Keep data forever to mine it for $$
  - Not support standards required by user (e.g., HIPAA, S&O)
- Malicious or buggy co-located user
  - Sniff side-channel information, degrade performance for others
- Hackers
  - Clouds are becoming "magnets" for sophisticated attacks
- Legal investigators
  - Force services to reveal data w/o user's knowledge or control

Review of Data Security/Privacy Challenges

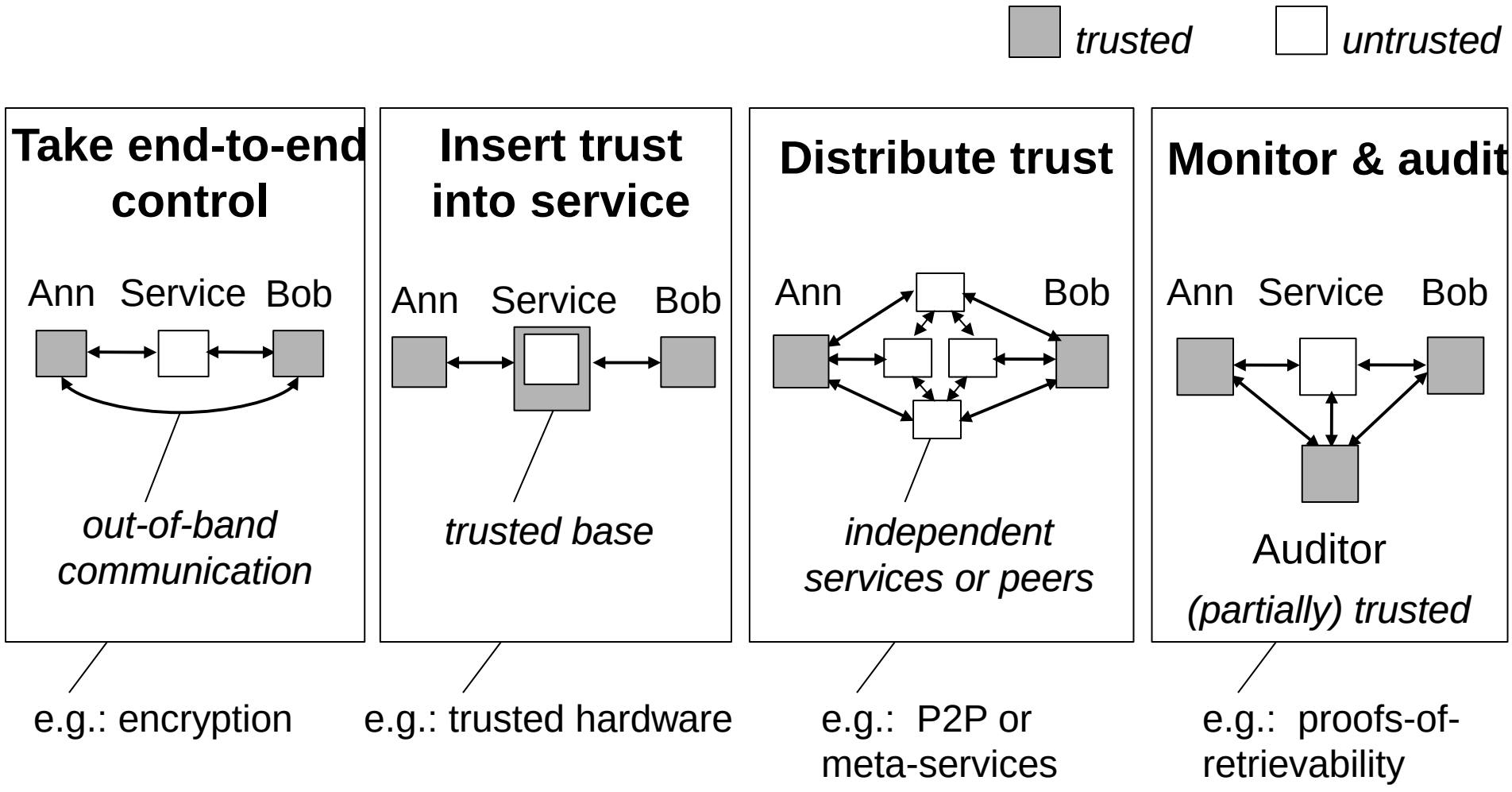
- Data confidentiality
  - How to keep data private when stored or computed on by cloud?
- Data access privacy
  - How to prevent Gmail from knowing whom you're emailing?
- Data access auditing
  - How to tell who has accessed your Facebook data?
- Data availability
  - How to ensure that your cloud data is replicated enough?
- Data persistence and integrity
  - How not to lose data when your cloud storage goes bankrupt?
- Data deletion
  - How to ensure all copies of your Gmail emails are deleted?

Untrusted Cloud Services: Threat Models

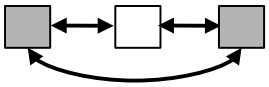
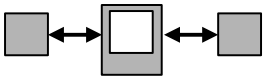


- Malicious (Byzantine) service:
  - Behaves arbitrarily due to economic / legal reasons or bugs
  - E.g.: releases or alters data, refuses deletion, etc.
- Trusted-until-proven-malicious service:
  - The user believes the service is trustworthy
  - Wishes to ensure detect when this assumption fails
    - E.g.: service is compromised or turns malicious

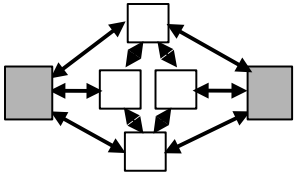
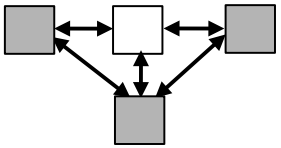
Four Ways for Dealing with Untrusted Services



Example Techniques (1/2)

Architecture		Example Techniques
End-to-end Control		- Encryption - Searchable encryption - Fully-homomorphic encryption - Private information retrieval - Attribute-based encryption -Trusted storage on untrusted clouds, e.g.: SPORC, SUNDR, Venus, CloudProof, CryptDB, …
Insert Trust		- TPMs - Terra isolated VMs - Distributed information flow control -Trusted incrementer - Attest-to-memory hardware

Example Techniques (2/2)

Architecture		Example Techniques
Distribute Trust		- Threshold secret sharing - Byzantine fault tolerance - Secure multi-party computations - …
Monitor & Audit		- Proofs-of-retrievability - Tamper-resistant logging - …

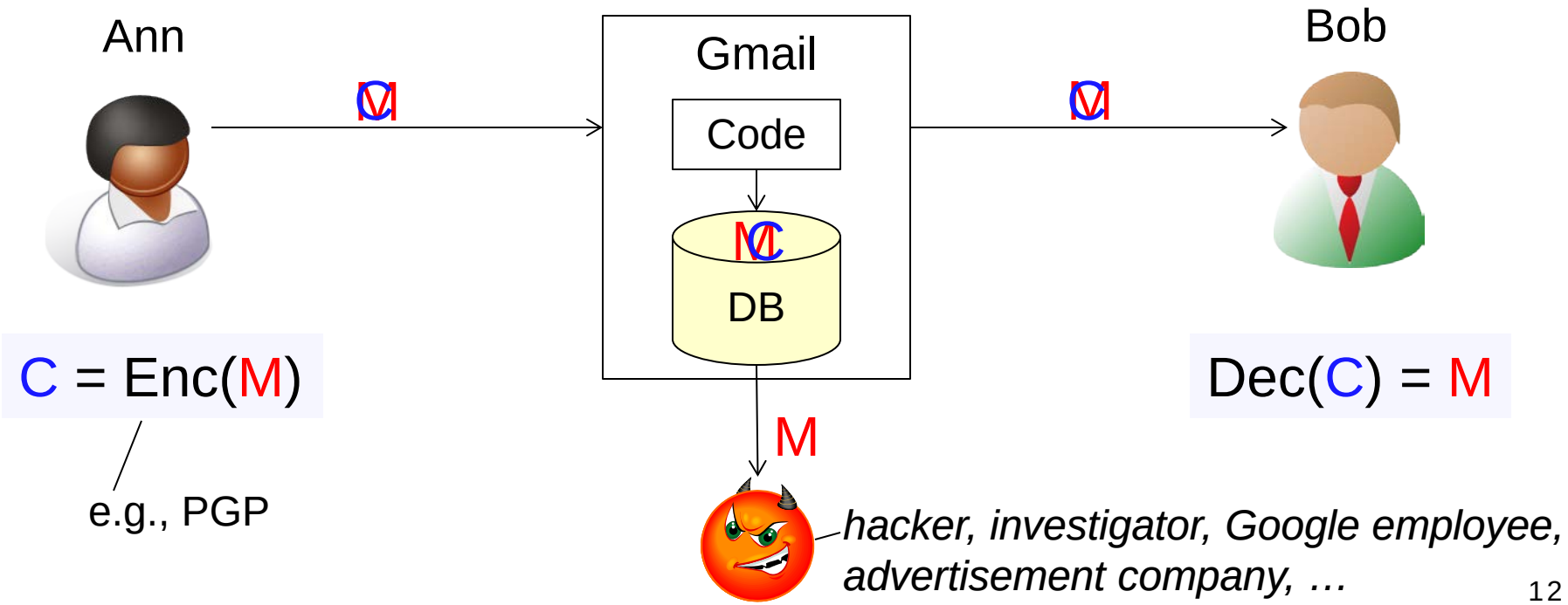
Today and the Next Time

- Encryption
- SQL-enabled encryption
- Fully-homomorphic encryption

- Important note:
  - This is a systems class, not a crypto class
  - We will talk about the semantics, application, and limitations of encryption techniques, not so much the math!
  - That's how you should also read the crypto papers

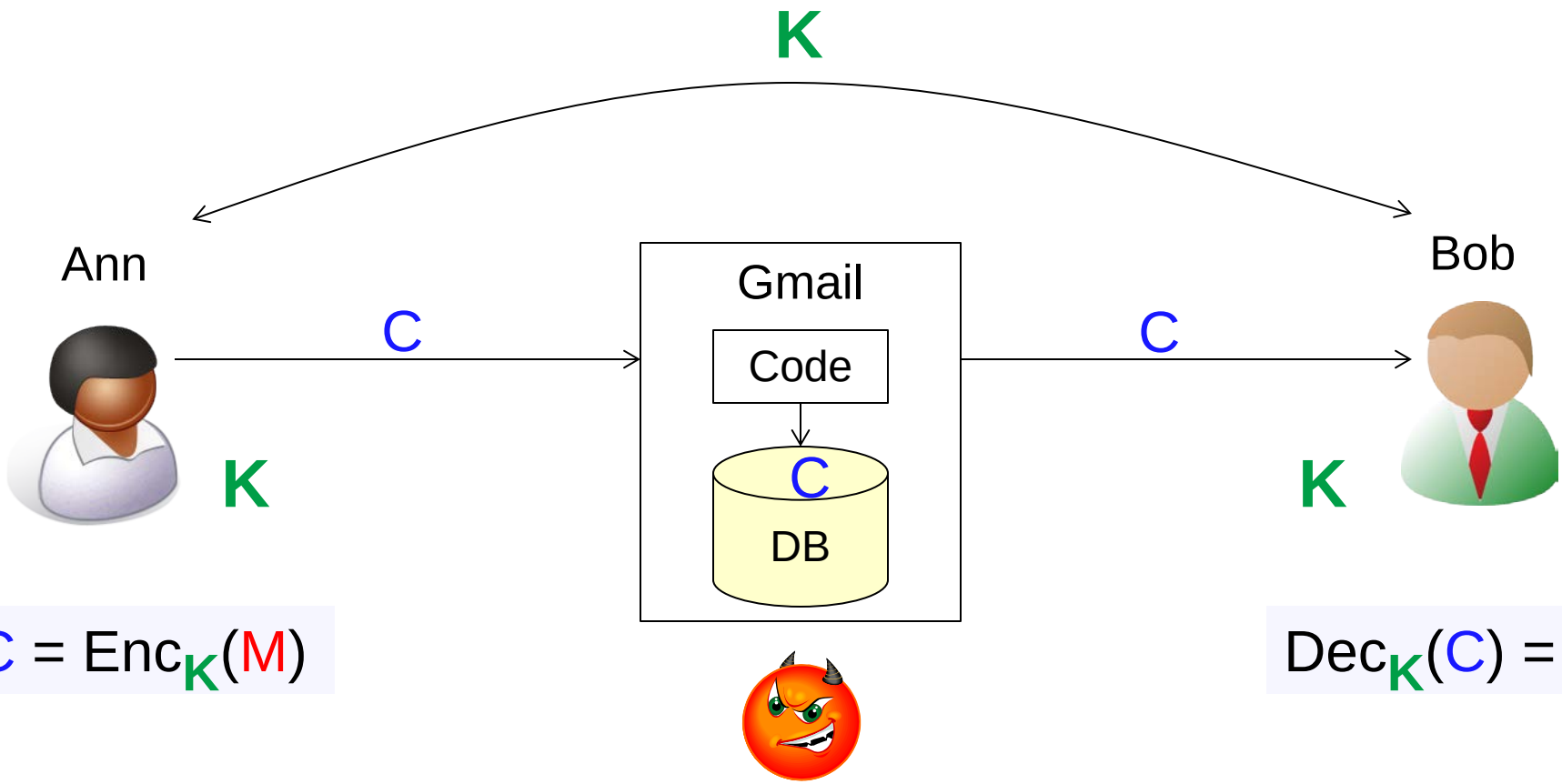
Encryption: Background

- Two traditional uses of encryption:
  - Protect data confidentiality
  - Protect data integrity
- Example: encrypt emails to prevent Gmail's eavesdropping



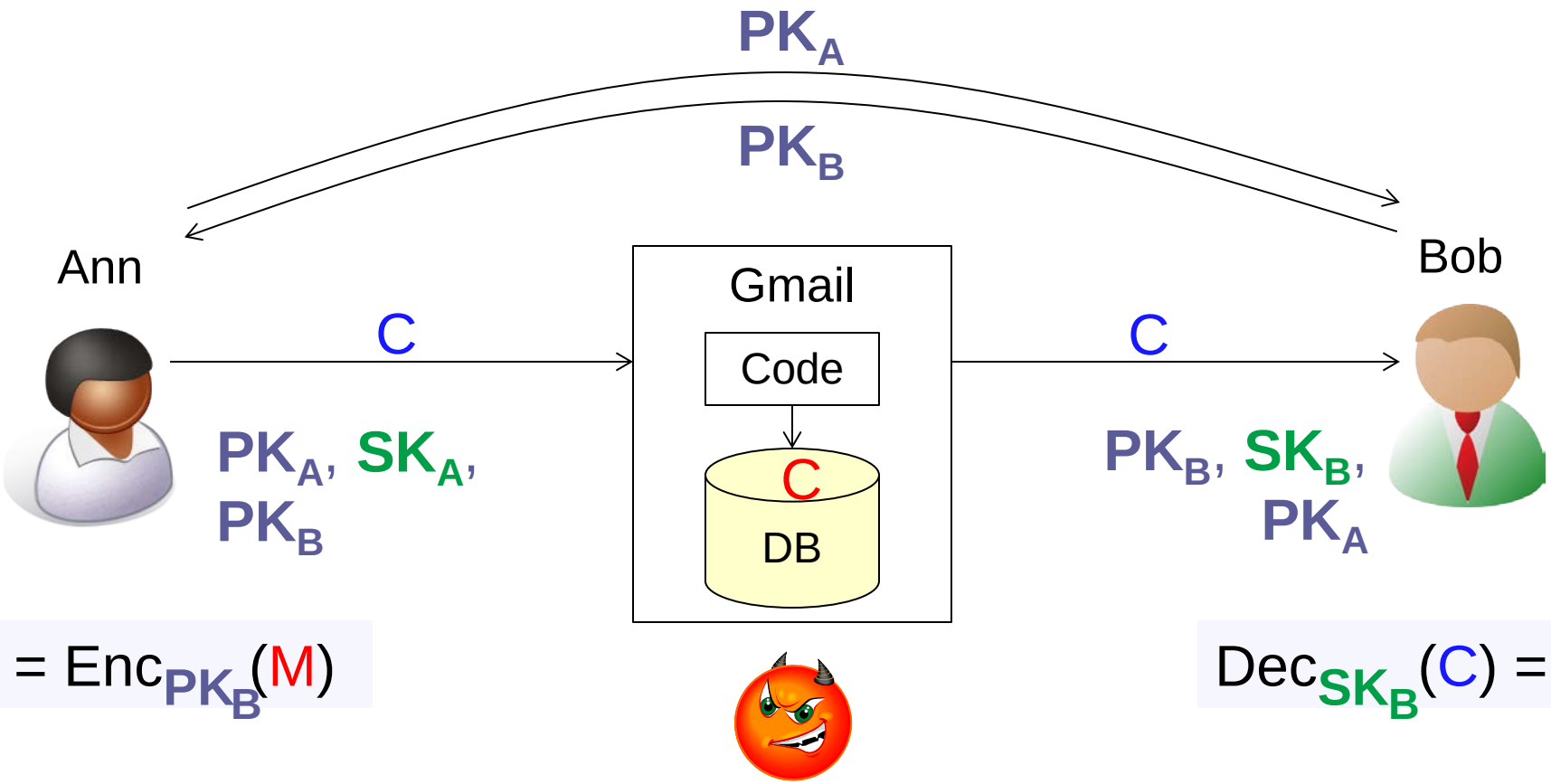
1. Symmetric Encryption (e.g., AES)

- Communicating parties share random string **K**, called the key
- **K** must be shared securely – very difficult!



2. Asymmetric Encryption (e.g., RSA)

- Each party creates a public key **PK** and a secret key **SK**
- Parties exchange **PK**'s, which don't need to be kept secret



Q: What other problems can encryption address in the cloud?

Encryption in the Cloud: Applications

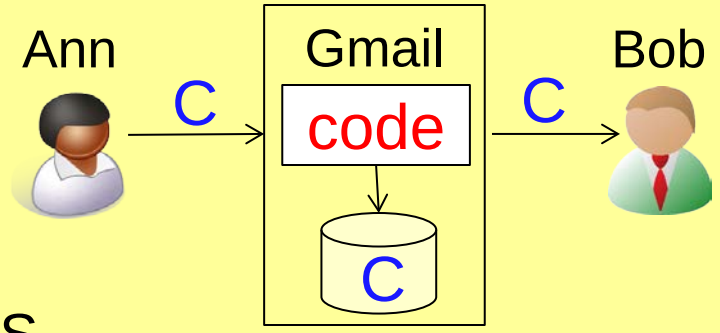
- Confidentiality
  - We've just seen example
- Integrity
  - Digital signatures based on public-key encryption
- Assured deletion
  - Ephemerizer [Perlman 05], Vanish [Geambasu, et.al. 09], CleanOS[Tang,et.al. 12] FADE [Tang, et.al. 10]
- Auditing
  - Keypad [Geambasu, et.al. 11]

Q: What challenges does traditional encryption raise in cloud?

Encryption in the Cloud: Challenges

- Standard: key management, performance, usability

- No service-side computations
  - Can't search in Gmail
  - Can't compute taxes in Netsuite
  - Can't operate on data in Amazon AWS
  - Can't get automatic face recognition on Facebook photos
  - …



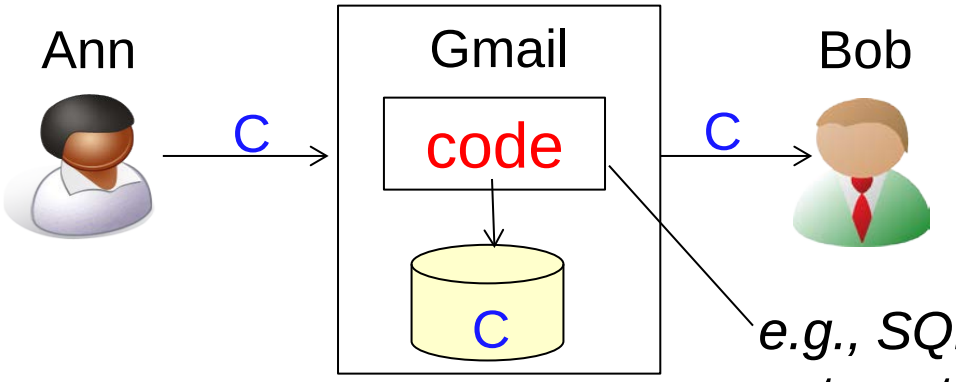
- Limited sharing
  - Compared to access control schemes like ACLs, capabilities

- Leaks access pattern information
  - E.g., whom user emails, which contacts' pages he visits, …

Computation-enabled Encryption

- Idea: - Let code run on encrypted data without decrypting it
  - Computation results are also encrypted

- Many special-case encryption schemes for popular functions
  - Keyword search, DB queries, certain arithmetic functions
- Recently, a general encryption scheme was designed that works for any function



e.g., SQL, indexing and search, automatic labeling, spam filtering, attachment virus detection, …

Outline

- Review of cloud challenges and solutions
- SQL-enabled encryption
  - **CryptDB: Protecting Confidentiality with Encrypted Query Processing.** Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. In *Proceedings of SOSP*, 2011.

Next time:

- Trusted collaborative cloud storage
- Fully-homomorphic encryption

CryptDB: Protecting Confidentiality with Encrypted Query Processing

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

MIT CSAIL

Presenters: Dewei Zhu, Shun-Xuan Wang

Current Situation & Solutions

- Sensitive information that stored on cloud is **NOT SECURE**.


a curious DBA

an adversary that gains complete control of applications and DBMS

- Solutions
  - Encrypt sensitive data and run all computations on clients.
    - BUT, several important applications do not lend themselves to this approach
  - Theoretic solutions such as full homomorphic encryption
    - prohibitively expensive

CryptDB

- A system that explores an intermediate design point to provide confidentiality for applications that use database management systems (DBMSes)
- Two threats model it addresses

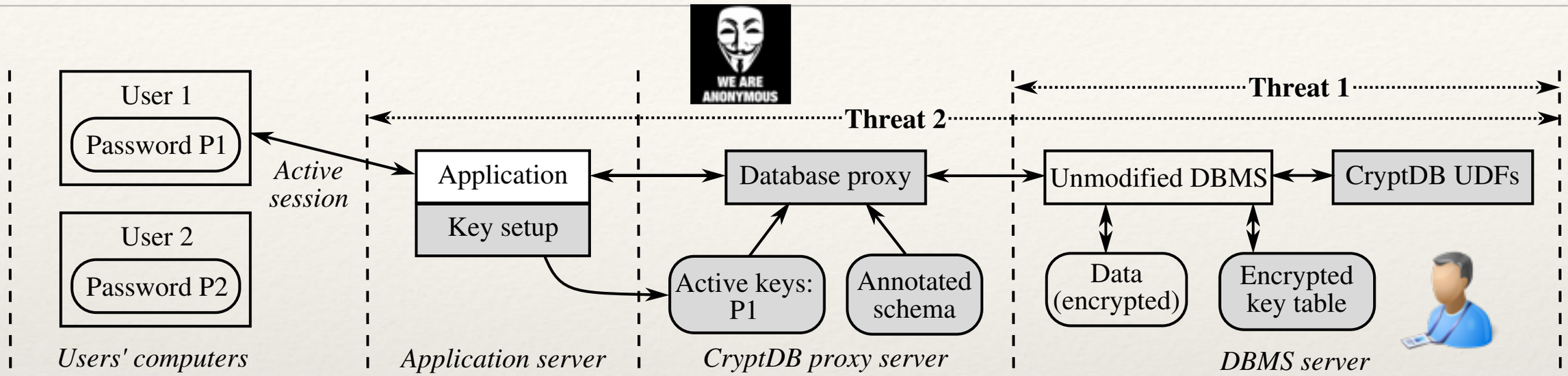
A curious database administrator (DBA) who tries to learn private data by snooping on the DBMS server

  - CryptDB prevents the DBA from learning private data

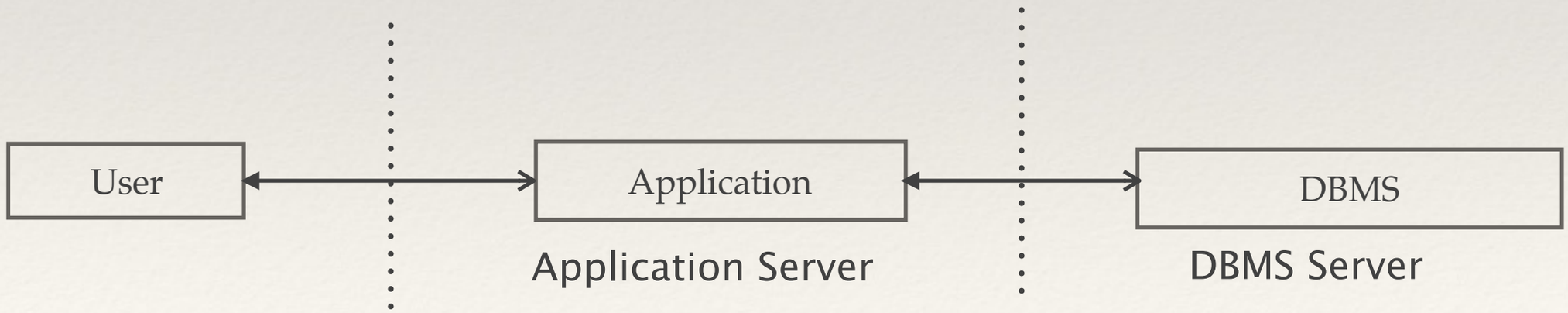
An adversary that gains complete control of application and DBMS servers

  - 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

CryptDB Architecture & Threat Models

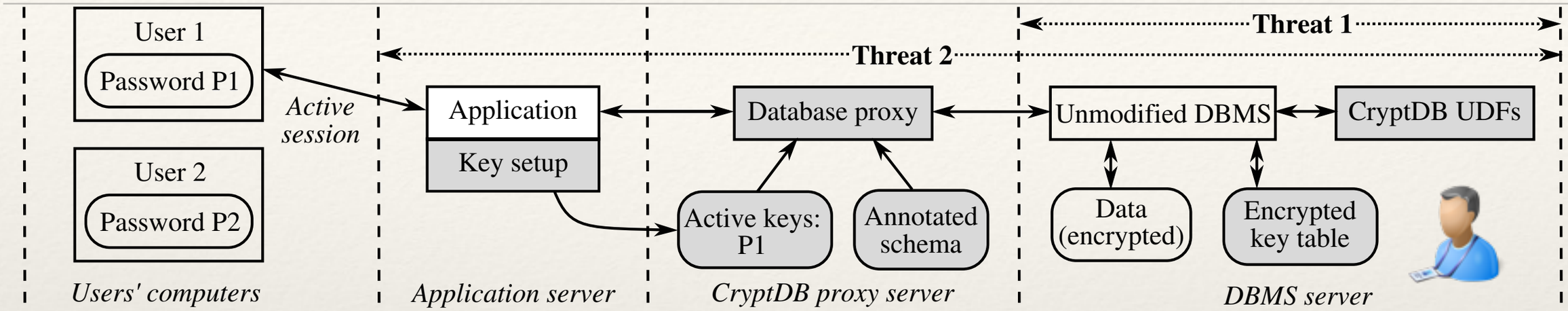


- Application Server
- CryptDB Proxy Server
- DBMS server



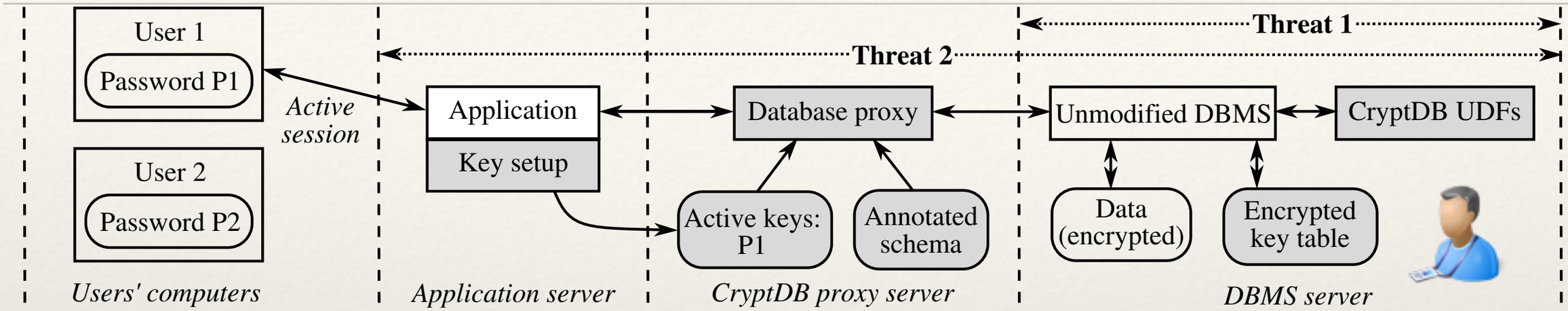
Normal Application Architecture

Threat 1: DBMS Server Compromise



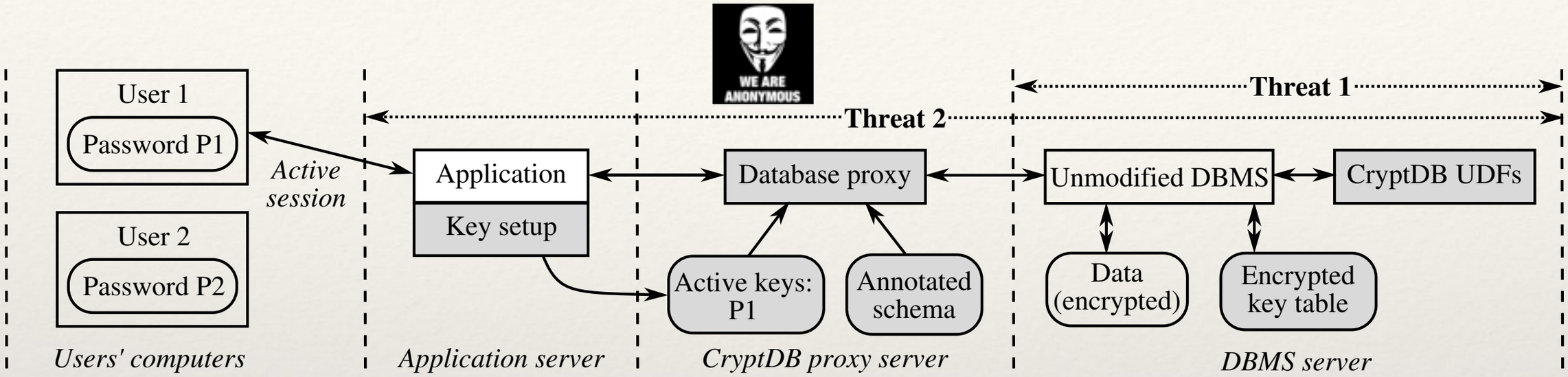
- This threat includes
  - DBMS software compromises
  - Root access to DBMS machines
  - Access to the RAM of physical machines
- Approach
  - Execute SQL queries over encrypted data on the DBMS server. (How to achieve this?)
- Guarantees
  - Confidentiality for data content and for names of columns and tables
  - Does not hide the overall table structure, the number of rows, the types of columns, or the approximate size of data in bytes.

Threat 1: DBMS Server Compromise



- The Security of CryptDB is not perfect
  - Reveals to the DBMS server relationships among data items that correspond to the classes of computation that queries perform on the database
- Assumption
  - The attacker is passive: she wants to learn confidential data, but does not change queries issued by the application, query results, or the data in the DBMS.
- Authors' Claim
  - A passive attack model is realistic because malicious DBAs are more likely to read the data, which may be hard to detect, than to change the data or query results, which is more likely to be discovered.

Threat 2: Arbitrary Threats

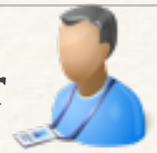


- Definition
  - The application server, proxy, and DBMS server infrastructures may be compromised arbitrarily.
- Solution
  - Encrypt different data items with different keys. To determine the key that should be used for each data item, developers annotate the ap- plication's database schema to express finer-grained confidentiality policies.

Discussion Topics

- Recall the definitions of both threat model
  - 1: DBMS software compromises, Root access to DBMS machines, Access to the RAM of physical machines
  - 2: The application server, proxy, and DBMS server infrastructures may be compromised arbitrarily.
- Is threat model 1 is a subset of the threat model 2? Is it necessary the author specialize one of the case of model 2 as threat model 1?

Queries Over Encrypted Data

- Threat Model: Compromised DBMS Server 
- Steps for processing a query in CryptDB
  - The application issues a query, which the proxy intercepts and rewrites: it anonymizes each table and column name, and, using the master key MK, encrypts each constant in the query with an encryption scheme best suited for the desired operation
  - The proxy checks if the DBMS server should be given keys to adjust encryption layers before executing the query, and if so, issues an UPDATE query at the DBMS server that invokes a UDF to adjust the encryption layer of the appropriate columns
  - The proxy forwards the encrypted query to the DBMS server, which executes it using standard SQL
  - The DBMS server returns the (encrypted) query result, which the proxy decrypts and returns to the application.

Different Encryption Types

- Motivation
  - Different operations in DBMS have different properties
- Random (RND)
  - Maximum security
  - Not efficient
  - Limited usage: used for select with out predicates
- Deterministic (DET)
  - Strong Security
  - If x = y, then DET(x) = DET(y)
  - Can perform selects with equality predicates, equality joins, GROUP BY, COUNT, DISTINCT, etc
- Order-preserving encryption (OPE).
  - Weaker than DET
  - If $x < y$, then $OPE_k(x) < OPE_k(y)$
  - Usage: ORDER BY, MIN, MAX, SORT

Different Encryption Types

- Homomorphic encryption (HOM)
  - Full homomorphic encryption is prohibitively slow
  - Implemented the Paillier crypto-system to support summation
  - $HOM_k(x) * HOM_k(y) = HOM_k(X*Y)$
  - Usage: SUM, AVG, +
- Join (JOIN and OPE-JOIN)
  - Usage: support all operations allowed by DET and enables the server to determine repeating values between two columns
  - Two kinds of Join operation: equi-join and range join
- Word search (SEARCH)
  - Need to call UDF
  - Usage: Support 'LIKE' Operation with full word search
  - Do not support regular expression

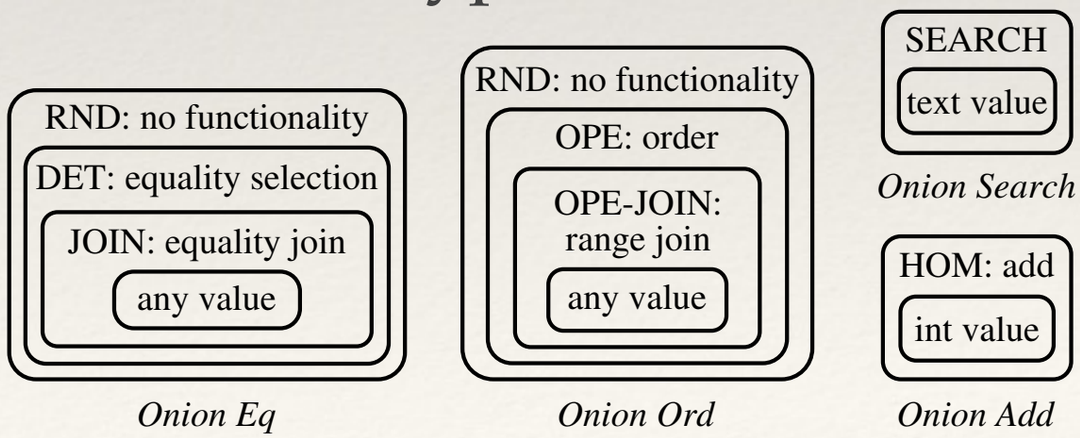
Computing Joins

- Two kinds of Join operations
  - equi-join: join predicate is based on equality
  - range-join: involve order check
- To perform equi-join
  - Two column should be encrypted by the same key
  - Two column that never ‘join’ should use different keys
- priori
  - the queries that can be issued
  - the pairs of columns that can be joined
- Problem
  - the proxy does not know the set of columns to be joined a priori and hence does not know which columns should be encrypted with matching keys
- Solution
  - Introduce a new cryptographic primitive, JOIN-ADJ (adjustable join), which allows the DBMS server to adjust the key of each column at runtime

Adjustable Query-based Encryption

- Description: dynamically adjusts the layer of encryption on the DBMS server
- Goal: Use the most secure encryption schemes that enable running the requested queries. (Greedy)

- Idea: Encrypt each data item in one or more onions
- Multiple onions are needed in practice.
  - The computations supported by different encryption schemes are not always strictly ordered
  - Performance considerations



Improving Security

- Minimum onion layers
  - Application developers can specify the lowest onion encryption layer that may be revealed to the server for a specific column
- In-proxy processing
  - Although CryptDB can evaluate a number of predicates on the server, evaluating them in the proxy can improve security by not revealing additional information to the server.
- Training mode.
  - CryptDB provides a training mode, which allows a developer to provide a trace of queries and get the resulting onion encryption layers for each field, along with a warning in case some query is not supported.
- Onion re-encryption
  - In cases when an application performs in- frequent queries requiring a low onion layer (e.g., OPE), CryptDB could be extended to re-encrypt onions back to a higher layer after the infrequent query finishes executing.

Optimizing Performance

- Developer annotations
  - Developer can provide explicit annotations indicating the sensitive fields and leave the remaining fields in plaintext.
- Known query set
  - Developer provides the exact query set, or annotations that certain functionality is not needed on some columns, CryptDB can also discard onions that are not needed
- Cipher-text pre-computing and caching
  - Proxy pre-computes (for HOM) and caches (for OPE) encryptions of frequently used constants under different keys

SQL Scheme Policy Annotations

- In SQL scheme, access privileges to data items of different groups can be specified.
- 1. Defining type of users: ***PRINCTYPE***
- 2. Defining which data to be encrypted: ***ENC_FOR***
- 3. Defining authorization & delegation: ***SPEAKS_FOR***

Policy Annotations

- The modified SQL scheme:
- ***PRINCTYPE***: who can access this item
- ***ENC_FOR***: which column should be encrypted
- ***SPEAKS_FOR***: ***username*** has all keys from ***userid***

```
PRINCTYPE physical_user EXTERNAL;
PRINCTYPE user, msg;

CREATE TABLE privmsgs (
  msgid    int,
  subject  varchar(255) ENC_FOR (msgid msg),
  msgtext  text         ENC_FOR (msgid msg) );

CREATE TABLE privmsgs_to (
  msgid int,   rcpt_id int,   sender_id int,
  (sender_id user) SPEAKS_FOR (msgid msg),
  (rcpt_id user)   SPEAKS_FOR (msgid msg) );

CREATE TABLE users (
  userid int,   username varchar(255),
  (username physical_user) SPEAKS_FOR (userid user) );
```

Key Chaining

- *B* speaks for *A*: *A*'s key is encrypted by *B*'s key
- Example: ***msg 5***'s key is encrypted by ***user 1***'s key, then encrypted by ***user 2***'s key separately.

Example table contents, without anonymized column names:

Table *privmsgs*

msgid	*subject*	*msgtext*
5	xcc82fa	x37a21f

Table *privmsgs_to*

msgid	*rcpt_id*	*sender_id*
5	1	2

Table *users*

userid	*username*
1	'Alice'
2	'Bob'

Key Chaining

- For external entities, assign random keys and encrypted by users' passwords to them.
- Question: How to track whether users are online?
- Maintaining an additional table to record users' logging in or out by ***INSERT*** or ***DELETE*** operations.

System Limitation

- CryptDB supports:
- rational queries, aggregates, mapping complex unsupported operations to simpler ones

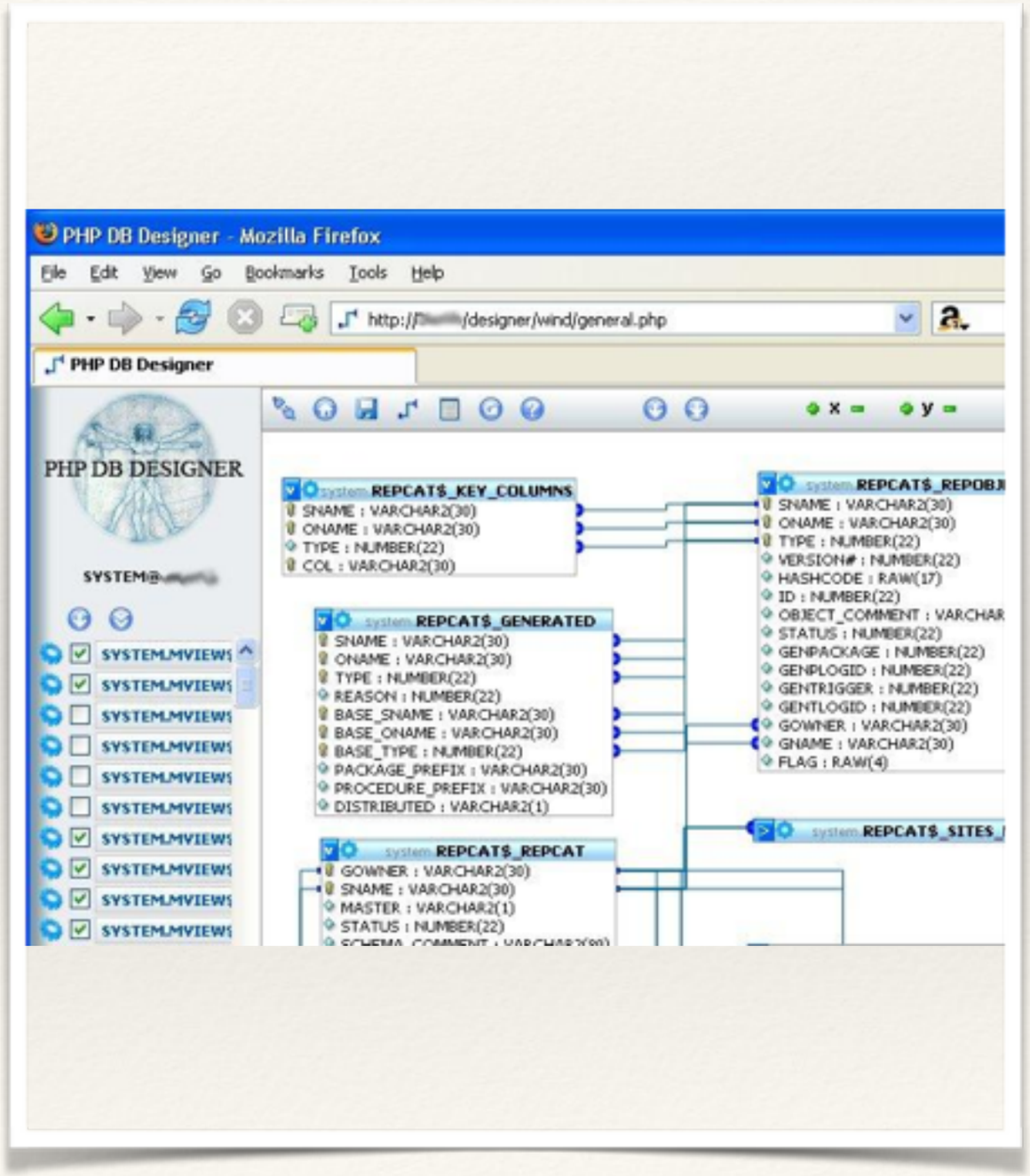
- CryptDB **NOT** supports:
- both computation and comparison on the same column
- For example: ***WHERE*** *salary > age*2+10*

Evaluation

- Evaluation metrics:
- 1. difficulties of modifying the application
- 2. types of queries and applications supported
- 3. level of security
- Applied CryptDB onto 3 different web applications: *phpDB, HotCRP,* and *MIT EECS-Application System*.

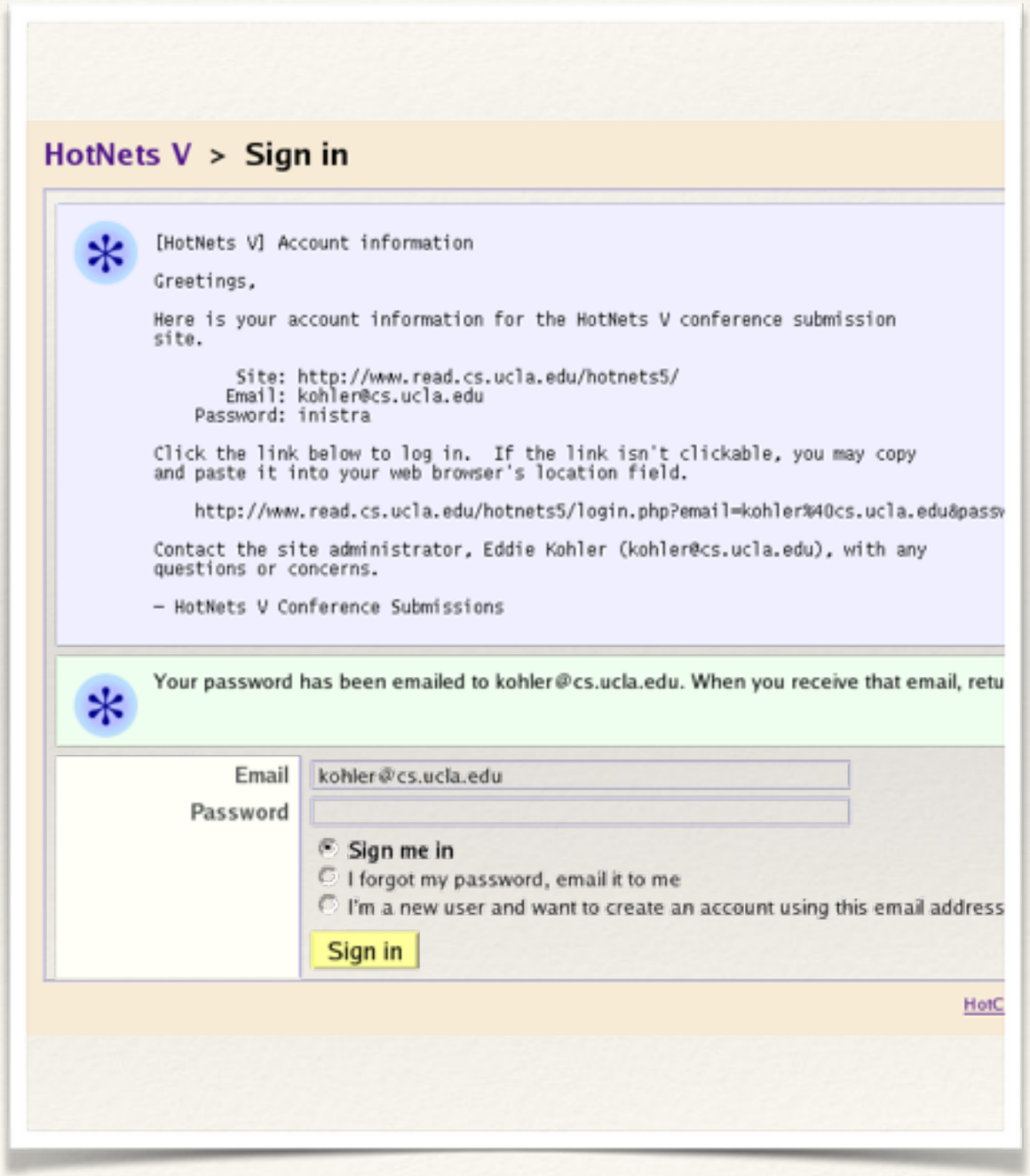
Evaluation: phpDB

- The Database that:
- Widely used open source forum with rich sets of access control settings.
- Supports varieties of access permissions.



Evaluation: HotCRP

- The Database about:
- Conference review application, as what we have done twice.
- PC members can not see who reviewed their papers.



Evaluation: grad-apply

- The Database about:
- Application website of MIT EECS.
- Applicants can see their data except for recommendations.
- Actually, it is a very user-unfriendly system……

Evaluation: Application Change

- The efforts required to use CryptDB in 3 web applications and single-principal TPC-C queries.
- 11–13 unique or 29 -111 total schema annotations to enforce privacy policies for up to 103 sensitive fields.
- 2 to 7 lines of source code changes.

Application	Annotations	Login/logout code	Sensitive fields secured, and examples of such fields
phpBB	31 (11 unique)	7 lines	23: private messages (content, subject), posts, forums
HotCRP	29 (12 unique)	2 lines	22: paper content and paper information, reviews
grad-apply	111 (13 unique)	2 lines	103: student grades (61), scores (17), recommendations, reviews
TPC-C (single princ.)	0	0	92: all the fields in all the tables encrypted

Evaluation: Functionality

- 571 out of 128,840 *in-proxy processing* are not supported. Therefore, overall 99.5% of columns are supported.
- Among these unsupported operations: 222 were bitwise operator, 205 were string processing, 76 were math transform, 41 were ***LIKE*** operator.

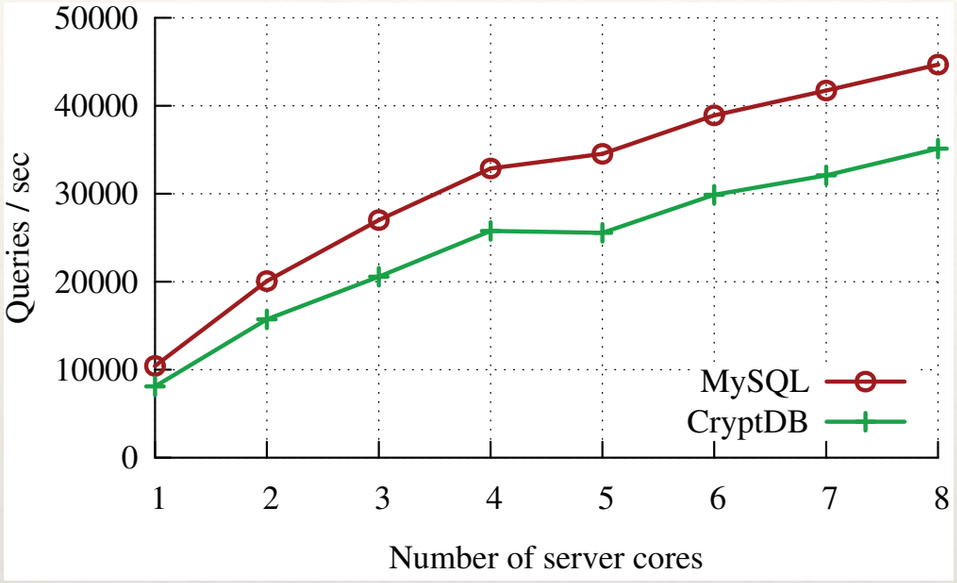
Application	**Total cols.**	**Consider for enc.**	**Needs plaintext**	**Needs HOM**	**Needs SEARCH**
phpBB	563	23	0	1	0
HotCRP	204	22	0	2	1
grad-apply	706	103	0	0	2
OpenEMR	1,297	566	7	0	3
MIT 6.02	15	13	0	0	0
PHP-calendar	25	12	2	0	2
TPC-C	92	92	0	8	0
Trace from `sql.mit.edu`	128,840	128,840	1,094	1,019	1,125
... with in-proxy processing	128,840	128,840	571	1,016	1,135
... col. name contains *pass*	2,029	2,029	2	0	0
... col. name contains *content*	2,521	2,521	0	0	52
... col. name contains *priv*	173	173	0	4	0

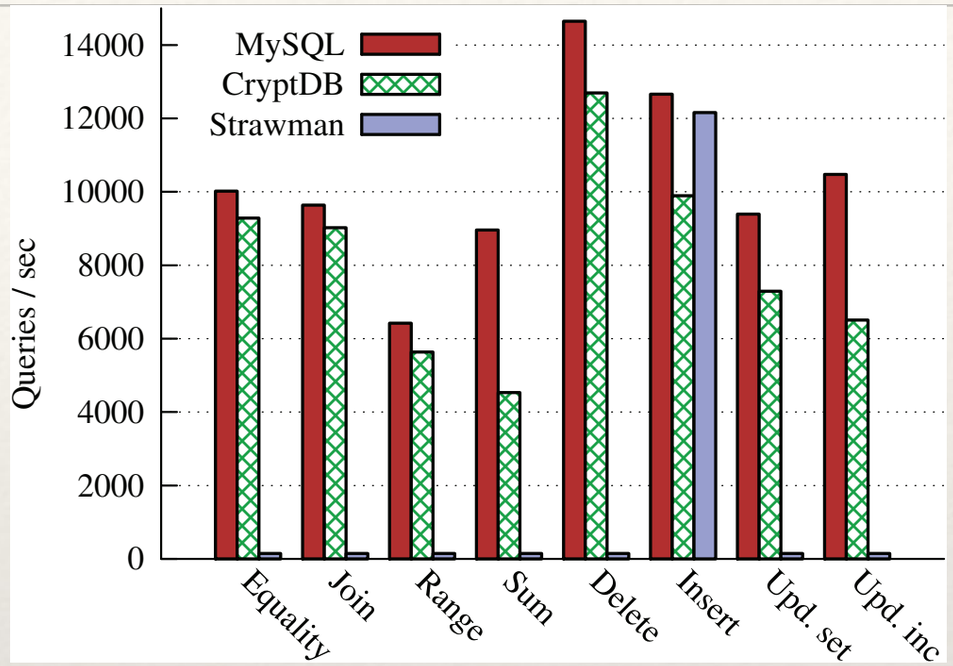
Evaluation: Security

- Security level: ***RND=HOM>SEARCH>DET=JOIN>OPE***
- About 6.6% were ***OPE***, 93% remain ***DET*** or above.

Application	Non-plaintext cols. with MinEnc:				Most sensitive cols. at HIGH
	RND	SEARCH	DET	OPE	
phpBB	21	0	1	1	6 / 6
HotCRP	18	1	1	2	18 / 18
grad-apply	95	0	6	2	94 / 94
OpenEMR	526	2	12	19	525 / 540
MIT 6.02	7	0	4	2	1 / 1
PHP-calendar	3	2	4	1	3 / 4
TPC-C	65	0	19	8	—
Trace from `sql.mit.edu`	80,053	350	34,212	13,131	—
. . . with in-proxy processing	84,008	398	35,350	8,513	—
. . . col. name contains *pass*	1,936	0	91	0	—
. . . col. name contains *content*	2,215	52	251	3	—
. . . col. name contains *priv*	159	0	12	2	—

Evaluation: Performance





- Compared with unmodified MySQL server:
- ***Throughput*** is 26% lower than MySQL.
- Strawman: decrypting all relevant data on each query and then re-encrypting the results.

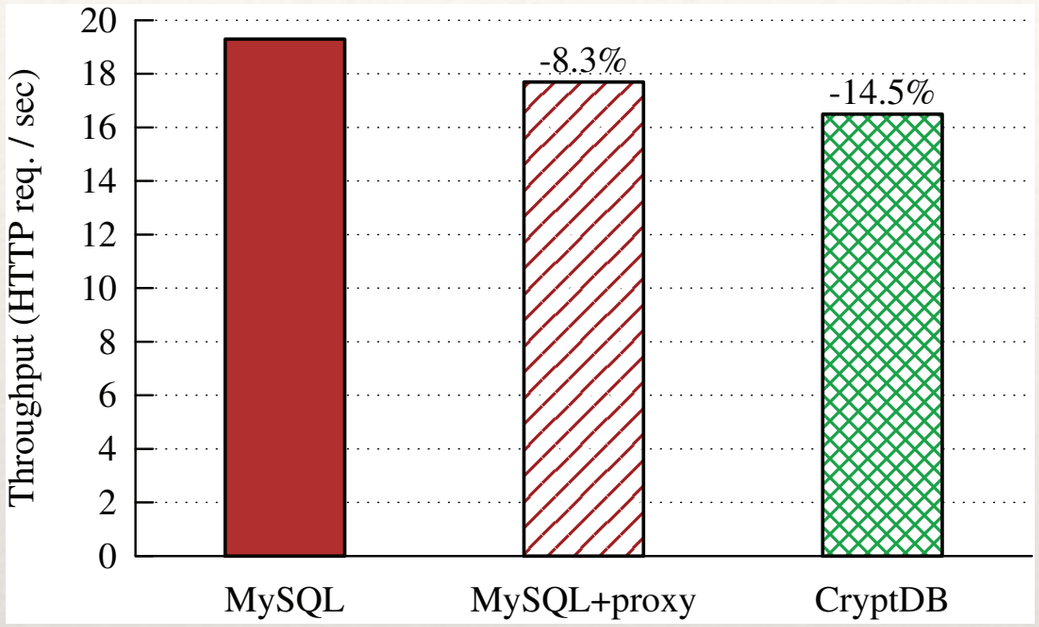
Evaluation: Performance

Query (& scheme)	MySQL Server	CryptDB		
		Server	Proxy	Proxy⋆
Select by = (DET)	0.10 ms	0.11 ms	0.86 ms	0.86 ms
Select join (JOIN)	0.10 ms	0.11 ms	0.75 ms	0.75 ms
Select range (OPE)	0.16 ms	0.22 ms	**0.78** ms	28.7 ms
Select sum (HOM)	0.11 ms	0.46 ms	0.99 ms	0.99 ms
Delete	0.07 ms	0.08 ms	0.28 ms	0.28 ms
Insert (all)	0.08 ms	0.10 ms	**0.37** ms	16.3 ms
Update set (all)	0.11 ms	0.14 ms	**0.36** ms	3.80 ms
Update inc (HOM)	0.10 ms	0.17 ms	**0.30** ms	25.1 ms
Overall	0.10 ms	0.12 ms	**0.60** ms	10.7 ms

Scheme	Encrypt	Decrypt	Special operation
Blowfish (1 int.)	0.0001 ms	0.0001 ms	—
AES-CBC (1 KB)	0.008 ms	0.007 ms	—
AES-CMC (1 KB)	0.016 ms	0.015 ms	—
OPE (1 int.)	9.0 ms	9.0 ms	Compare: 0 ms
SEARCH (1 word)	0.01 ms	0.004 ms	Match: 0.001 ms
HOM (1 int.)	9.7 ms	0.7 ms	Add: 0.005 ms
JOIN-ADJ (1 int.)	0.52 ms	—	Adjust: 0.56 ms

- ***Latency*** increased 20% by CryptDB
- Added 0.6 ms on average, 24% from MySQL proxy, 23% from encrypting/decrypting, 53% from parsing queries.

Evaluation: Multi-User



DB	Login	R post	W post	R msg	W msg
MySQL	60 ms	50 ms	133 ms	61 ms	237 ms
CryptDB	67 ms	60 ms	151 ms	73 ms	251 ms

- With 10 parallel clients, the 14.5% of ***throughput*** losses by CryptDB.

Extensions (1)

- Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. ***Processing Analytical Queries over Encrypted Data***. In Proceedings of the 39th International Conference on Very Large Data Bases (VLDB), 2013.
- Extends CryptDB to support complex analytical queries.
- Tests CryptDB with a large dataset.

input : Q, an abstract syntax tree (AST) for the query.
E, a description of the available encrypted columns.
output : P, a plan node for query, such as the one in Figure 3.

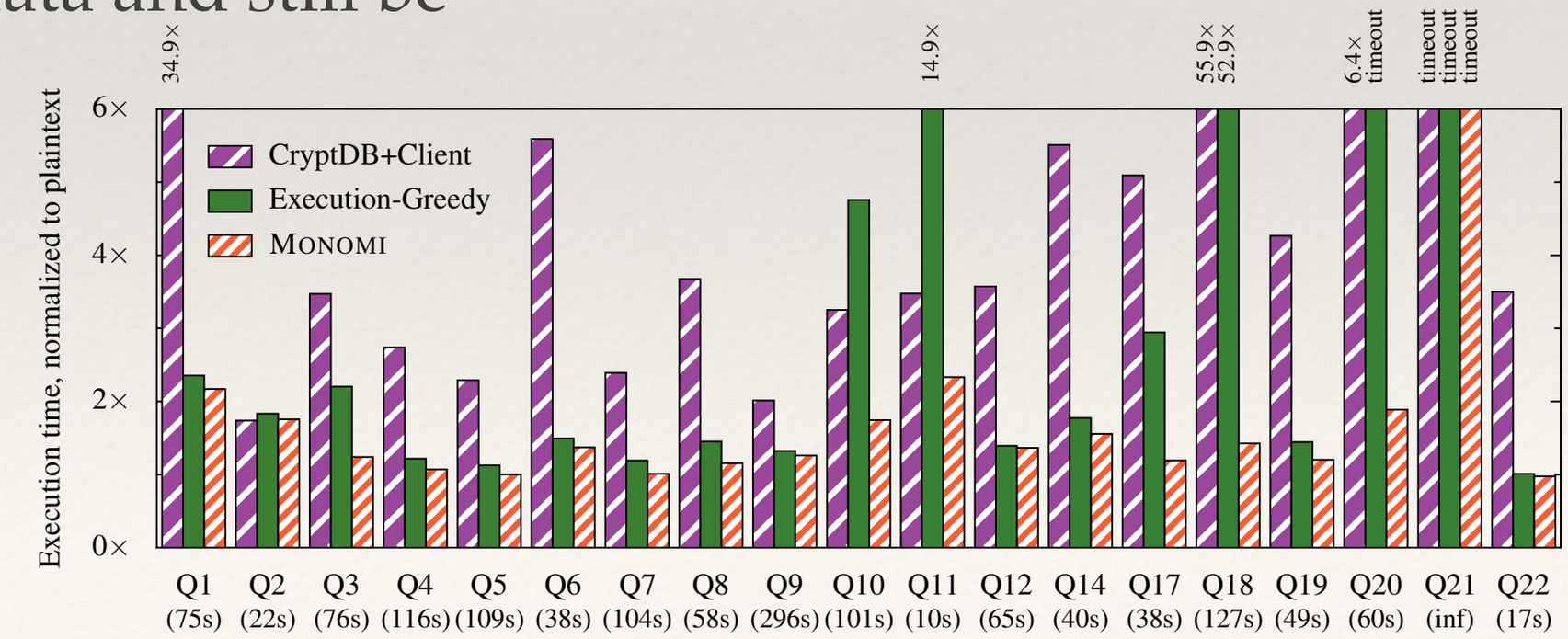
```
1  for s in subqueries in Q.relations do
2      p ← GENERATEQUERYPLAN(Q, E)
3      Replace s with p in Q.relations
4  RemoteQ ← Q                          // Query to run over encrypted data
5  LocalFilters ← []
6  for c in Q.join_clauses || [Q.where_clause] do
7      c' ← REWRITESERVER(c, E, PLAIN)
8      if c' ≠ Nil then                 // c computable on server
9          Replace c with c' in RemoteQ
10     else
11         Remove c from RemoteQ
12         Add c to LocalFilters
13         Add EXPRS(c) to RemoteQ.projections
14 RemoteQ.group_by.keys ← []
15 for k in Q.group_by.keys do
16     k' ← REWRITESERVER(k, E, DET)
17     if k' ≠ Nil then
18         Add k' to RemoteQ.group_by.keys
19 LocalGroupBy ← Nil
20 LocalHaving ← Nil
21 if RemoteQ.group_by.keys.size = Q.group_by.keys.size then
                                        // GROUP BY pushed onto server
22     f' ← REWRITESERVER(Q.group_by.having, E, PLAIN)
23     RemoteQ.group_by.having ← f'
24     if f' = Nil and Q.group_by.having ≠ Nil then
25         LocalHaving ← Q.group_by.having
26         Add EXPRS(LocalHaving) to RemoteQ.projections
27 else                                 // Compute GROUP BY on client
28     RemoteQ.group_by ← Nil
```

Extensions (1)

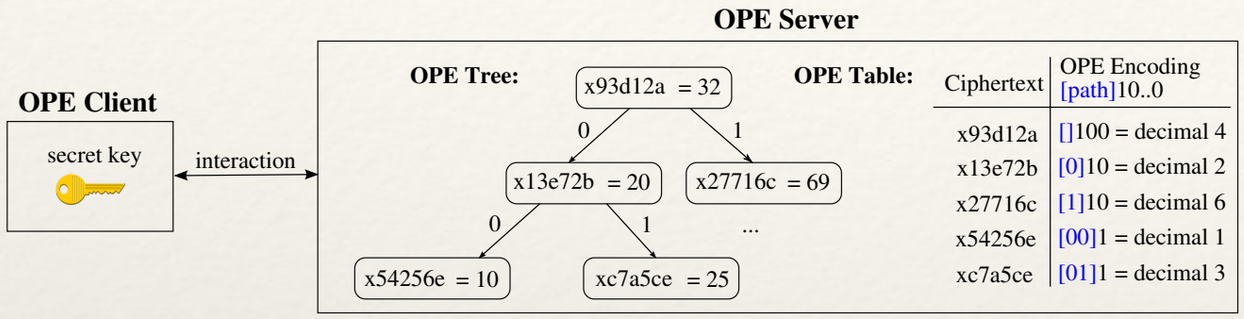
- Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. ***Processing Analytical Queries over Encrypted Data***.

System	Size (GB)	Relative to plaintext
Plaintext	17.10	–
CryptDB+Client	71.98	4.21×
Execution-Greedy	32.55	1.90×
MONOMI	29.38	1.72×

- **A Discussion Question:**
  Could it be used to store a very large amount of data and still be efficient enough?



Extensions (2)



- An encrypted binary search tree.
- Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich. ***An Ideal-Security Protocol for Order-Preserving Encoding***. In Proceedings of the 34th IEEE Symposium on Security and Privacy (IEEE S&P/Oakland), 2013.
- Extends CryptDB to support order-preserving queries.

Algorithm 1 (OPE Tree traversal for a value v)**.**

1: Cl $\leftrightarrow$ Ser: The client asks the server for the ciphertext c' at the root of the OPE tree.
2: Cl: The client decrypts c' and obtains v'.
3: Cl $\rightarrow$ Ser: If $v < v'$, the client tells the server "left"; if $v = v'$ the client tells the server "found"; if $v > v'$, the client tells the server "right".
4: Ser: The server returns the next ciphertext c'' based on the client's information, and goes back to step 2.
5: Ser, Cl: The algorithm stops when v is found, or when the server arrives at an empty spot in the tree. The outcome of the algorithm is the resulting pointer in the OPE Tree, the path in the OPE Tree from the root, and whether v was found.

Discussion

- Even though the security problem is released by CryptDB but the cost on it cannot be ignored. Considering the business benefit only, will the reward from the increased security cover the cost on the system?
- What percentage of developers actually modify a large framework which they customized for their needs?
- Somebody would like to discuss how Google's and MIT LL's implementations differ.

Discussion

- Is there any way to remove the proxy from this architecture but not end up using private/public key cryptography for everything?
- Could CryptDB use its training ability in order to find the optimal interval for re-encrypting the columns? If not (say because traffic and queries are constantly changing), is there any other methodology to use (e.g. re-encrypt if all columns are at the minimum level)?

Conclusion

- CryptDB is an encrypted database system which is:
- 1. practical and confidential,
- 2. running queries over encrypted data,
- 3. dynamically adjusting encryption level,
- 4. minimized information leakage on untrusted server.