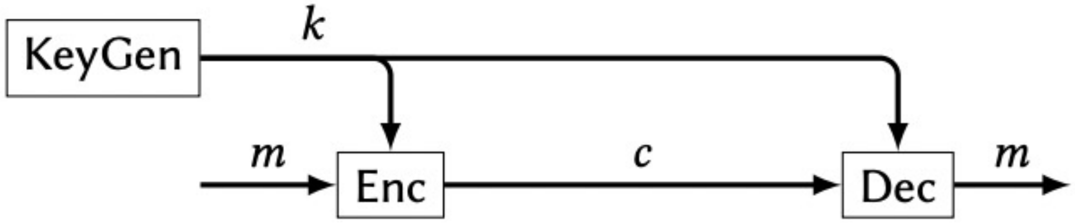
CS848
Background: crypto and db basics

Sujaya Maiyya

One-time pad



- Let $\{0, 1\}^{\lambda}$ denote the set of λ-*bit* binary strings

KeyGen:	$\text{Enc}(k, m \in \{0,1\}^{\lambda})$:	$\text{Dec}(k, c \in \{0,1\}^{\lambda})$:
$k \leftarrow \{0,1\}^{\lambda}$ return k	return $k \oplus m$	return $k \oplus c$

A ‘good’ encryption scheme

“an encryption scheme is a good one if its ciphertexts look like random junk to an attacker”

Let CTXT be a function callable by an adversary who can choose *m* and who sees *c*

CTXT(m):	
$k \leftarrow \{0,1\}^\lambda$	// KeyGen *of OTP*
$c := k \oplus m$	// Enc *of OTP*
return c	

vs.

CTXT(m):	
$c \leftarrow \{0,1\}^\lambda$	// C *of OTP*
return c	

Real vs Random

“an encryption scheme is a good one if encryptions of m_L look like encryptions of m_R to an attacker”

Let EAVESDROP be a function callable by an adversary who chooses m_L & m_R and sees c

EAVESDROP(m_L, m_R):	
$k \leftarrow \{0,1\}^\lambda$	// KeyGen *of OTP*
$c := k \oplus m_L$	// Enc *of OTP*
return c	

EAVESDROP(m_L, m_R):	
$k \leftarrow \{0,1\}^\lambda$	// KeyGen *of OTP*
$c := k \oplus m_R$	// Enc *of OTP*
return c	

Left vs right

Basics of provable security: Interchangeability

Let *L* be a library of functions that make random choices and let *A* be a calling program (callable by an adversary)

$\mathcal{L}$
$\underline{\textsc{ctxt}(m)}$:
$k \leftarrow \{0,1\}^\lambda$
$c := k \oplus m$
return c

$\mathcal{A}$:
$m \leftarrow \{0,1\}^\lambda$
$c := \textsc{ctxt}(m)$
return $m \overset{?}{=} c$

then $\Pr[\mathcal{A} \diamond \mathcal{L} \Rightarrow \mathtt{true}] = 1/2^\lambda$

Let $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ be two libraries that have the same interface. We say that $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ are ***interchangeable****, and write $\mathcal{L}_{\text{left}} \equiv \mathcal{L}_{\text{right}}$, if for all programs $\mathcal{A}$ that output a boolean value,*

$$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow \mathtt{true}] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow \mathtt{true}].$$

Basics of provable security: proving 'insecurity'

For OTP (and $\mathcal{A}$ from last slide) , $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow true] = \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow true] = 1/2^{\lambda}$

$\mathcal{L}_{\text{otp-real}}$

EAVESDROP$(m \in \{0,1\}^{\lambda})$:
- $k \leftarrow \{0,1\}^{\lambda}$ // OTP.KeyGen
- return $k \oplus m$ // OTP.Enc(k, m)

$\equiv$

$\mathcal{L}_{\text{otp-rand}}$

EAVESDROP$(m \in \{0,1\}^{\lambda})$:
- $c \leftarrow \{0,1\}^{\lambda}$ // OTP.C
- return c

Just one calling program enough to prove insecurity

What about??

$\mathcal{L}^{\Sigma}_{\text{ots\$-real}}$

CTXT$(m \in \{0,1\}^{\lambda})$:
- $k \leftarrow \{0,1\}^{\lambda}$ // Σ.KeyGen
- $c := k \,\&\, m$ // Σ.Enc
- return c

$\mathcal{L}^{\Sigma}_{\text{ots\$-rand}}$

CTXT$(m \in \{0,1\}^{\lambda})$:
- $c \leftarrow \{0,1\}^{\lambda}$ // Σ.C
- return c

$\mathcal{A}$:

- $c :=$ CTXT(0^{λ})
- return $c \stackrel{?}{=} 0^{\lambda}$

$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots\$-real}} \Rightarrow \texttt{true}] = 1$ $\quad$ $\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{ots\$-rand}} \Rightarrow \texttt{true}] = 1/2^{\lambda}$

Indistinguishability

Two libraries $\mathcal{L}_{left}$ *and* $\mathcal{L}_{right}$ are *indistinguishable* if for all polynomial-time calling programs $\mathcal{A}$,

$$\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] \approx \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1] \quad \rightarrow \quad \mathcal{L}_{\text{left}} \approxeq \mathcal{L}_{\text{right}}$$

Advantage or bias of an adversary: $\left|\Pr[\mathcal{A} \diamond \mathcal{L}_{\text{left}} \Rightarrow 1] - \Pr[\mathcal{A} \diamond \mathcal{L}_{\text{right}} \Rightarrow 1]\right|$

Two libraries are indistinguishable if the adversary's advantage is negligible (some small number like $1/2^{128}$)

Pseudorandom generators

- A pseudorandom generator (PRG) is a deterministic function G whose outputs are longer than its inputs

Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+\ell}$ *be a deterministic function with* $\ell > 0$. *We say that* G *is a* ***secure pseudorandom generator (PRG)*** *if* $\mathcal{L}^G_{\text{prg-real}} \approx \mathcal{L}^G_{\text{prg-rand}}$, *where:*

$\mathcal{L}^G_{\text{prg-real}}$
$\underline{\textsc{query}():}$ $s \leftarrow \{0,1\}^\lambda$ return $G(s)$

$\mathcal{L}^G_{\text{prg-rand}}$
$\underline{\textsc{query}():}$ $r \leftarrow \{0,1\}^{\lambda+\ell}$ return r

The value ℓ *is called the* ***stretch*** *of the PRG. The input to the PRG is typically called a* ***seed***.

Pseudorandom Functions (PRFs)

- PRFs are functions of the form

```
for x ∈ {0,1}^in:
  T[x] ← {0,1}^out

LOOKUP(x ∈ {0,1}^in):
  return T[x]
```

Let $F : \{0,1\}^\lambda \times \{0,1\}^{in} \to \{0,1\}^{out}$ be a deterministic function. We say that F is a secure ***pseudorandom function (PRF)*** *if $\mathcal{L}^F_{\text{prf-real}} \approx \mathcal{L}^F_{\text{prf-rand}}$, where:*

$\mathcal{L}^F_{\text{prf-real}}$

```
k ← {0,1}^λ

LOOKUP(x ∈ {0,1}^in):
  return F(k, x)
```

$\mathcal{L}^F_{\text{prf-rand}}$

```
T := empty assoc. array

LOOKUP(x ∈ {0,1}^in):
  if T[x] undefined:
    T[x] ← {0,1}^out
  return T[x]
```

Pseudorandom Permutations (PRPs) or block ciphers

- Let K be the keyspace, X the message or input space and Y the output space.
- A PRF, F:

$$F : K \times X \rightarrow Y$$

- A PRP, E :

$$E : K \times X \rightarrow X$$

- A PRP is required to be bijective, and to have an efficient inversion function, PRP^{-1}.
- E is also called block cipher: E corresponds to encryption, E^{-1} corresponds to decryption, and all outputs of E look pseudorandom

Security Against Chosen Plaintext Attacks

Let Σ be an encryption scheme. We say that Σ has ***security against chosen-plaintext attacks (CPA security)*** *if $\mathcal{L}^{\Sigma}_{\text{cpa-L}} \approx \mathcal{L}^{\Sigma}_{\text{cpa-R}}$, where:*

$\mathcal{L}^{\Sigma}_{\text{cpa-L}}$	$\mathcal{L}^{\Sigma}_{\text{cpa-R}}$
$k \leftarrow \Sigma.\text{KeyGen}$	$k \leftarrow \Sigma.\text{KeyGen}$
$\textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M})$:	$\textsc{eavesdrop}(m_L, m_R \in \Sigma.\mathcal{M})$:
$c := \Sigma.\text{Enc}(k, m_L)$	$c := \Sigma.\text{Enc}(k, m_R)$
return c	return c

Deterministic encryption like block ciphers are not CPA-secure!

One simple (nonce-based) randomized encryption scheme using PRFs

Let F be a secure PRF with in $= \lambda$. Define the following encryption scheme based on F:

$$\mathcal{K} = \{0,1\}^\lambda$$
$$\mathcal{M} = \{0,1\}^{out}$$
$$\mathcal{C} = \{0,1\}^\lambda \times \{0,1\}^{out}$$

KeyGen:
- $k \leftarrow \{0,1\}^\lambda$
- return k

Enc(k, m):
- $r \leftarrow \{0,1\}^\lambda$
- $x := F(k, r) \oplus m$
- return (r, x)

Dec$(k, (r, x))$:
- $m := F(k, r) \oplus x$
- return m

This scheme is also ‘**symmetric**’: same secret key is used for encryption and decryption

Public-key or asymmetric encryption

- Goal: make encryption key public, so that anyone can send an encryption to the owner of that key, even if the two users have never spoken before and have no shared secrets.

- The decryption key is private, so that only the designated owner can decrypt.

Public-key or asymmetric encryption

Let Σ be a public-key encryption scheme. Then Σ is **secure against chosen-plaintext attacks (CPA secure)** *if $\mathcal{L}^{\Sigma}_{\text{pk-cpa-L}} \approx \mathcal{L}^{\Sigma}_{\text{pk-cpa-R}}$, where:*

$\mathcal{L}^{\Sigma}_{\text{pk-cpa-L}}$
$(pk, sk) \leftarrow \Sigma.\text{KeyGen}$
$\underline{\text{GETPK}():}$
$\quad$ return pk
$\underline{\text{CHALLENGE}(m_L, m_R \in \Sigma.\mathcal{M}):}$
$\quad$ return $\Sigma.\text{Enc}(pk, m_L)$

$\mathcal{L}^{\Sigma}_{\text{pk-cpa-R}}$
$(pk, sk) \leftarrow \Sigma.\text{KeyGen}$
$\underline{\text{GETPK}():}$
$\quad$ return pk
$\underline{\text{CHALLENGE}(m_L, m_R \in \Sigma.\mathcal{M}):}$
$\quad$ return $\Sigma.\text{Enc}(pk, m_R)$

Example – ElGamal encryption

Let *m* be the message we want to encrypt

Encryption key: (s, p) *such that* $p = g^s$

Encryption procedure:

$E(p, m)$:

Pick a random number r

Return $(g^r , m * p^r)$

Call it A

Call it B

Decryption procedure:

$D(sk, E(p,m)) = B\ (A^{sk})^{-1}$

$= D(s, (g^r , m * p^r))$

$= (m*p^r) * ((g^r)^s)^{-1}$

$= (m * (g^s)^r * (g^{rs})^{-1}$

$= m$

Homomorphic encryption

- Allows computing – addition and multiplication – on encrypted data and result is also encrypted

ElGamal encryption is partially homomorphic – supports homomorphic **multiplication**

p is public key

$$E(m1) * E(m2) = (g^{r1}, m1 * p^{r1}) * (g^{r2}, m2 * p^{r2})$$

$$= (g^{r1+r2}, (m1 * m2) * p^{r1+r2})$$

$$= E(m1 * m2)$$

Homomorphic encryption (additive)

Modified El Gamal encryption procedure:

E(p, m):

Pick a random number *r*

Return $(g^r, g^m * p^r)$

This version supports homomorphic additions

p is public key

$$E(m1) * E(m2) = (g^{r1}, g^{m1} * p^{r1}) * (g^{r2}, g^{m2} * p^{r2})$$

$$= (g^{r1 + r2},\ g^{(m1 + m2)} * p^{r1 + r2})$$

$$= E(m1 + m2)$$

Order preserving encryption (OPE)

- If in plaintext, $x < y$, then encrypting these values using an OPE ensures that for any secret key k,

$$OPE_k(x) < OPE_k(y)$$

- OPEs help serve range queries on encrypted data but it leaks more information than non-deterministic encryption schemes

Database basics

Basic db queries

- Simple key-value stores support single key GET and PUT queries

More complex queries

- Selection

  SELECT * from T1 where age = 20

- Range

  Select name, salary from T1 where age > 20 and age < 60

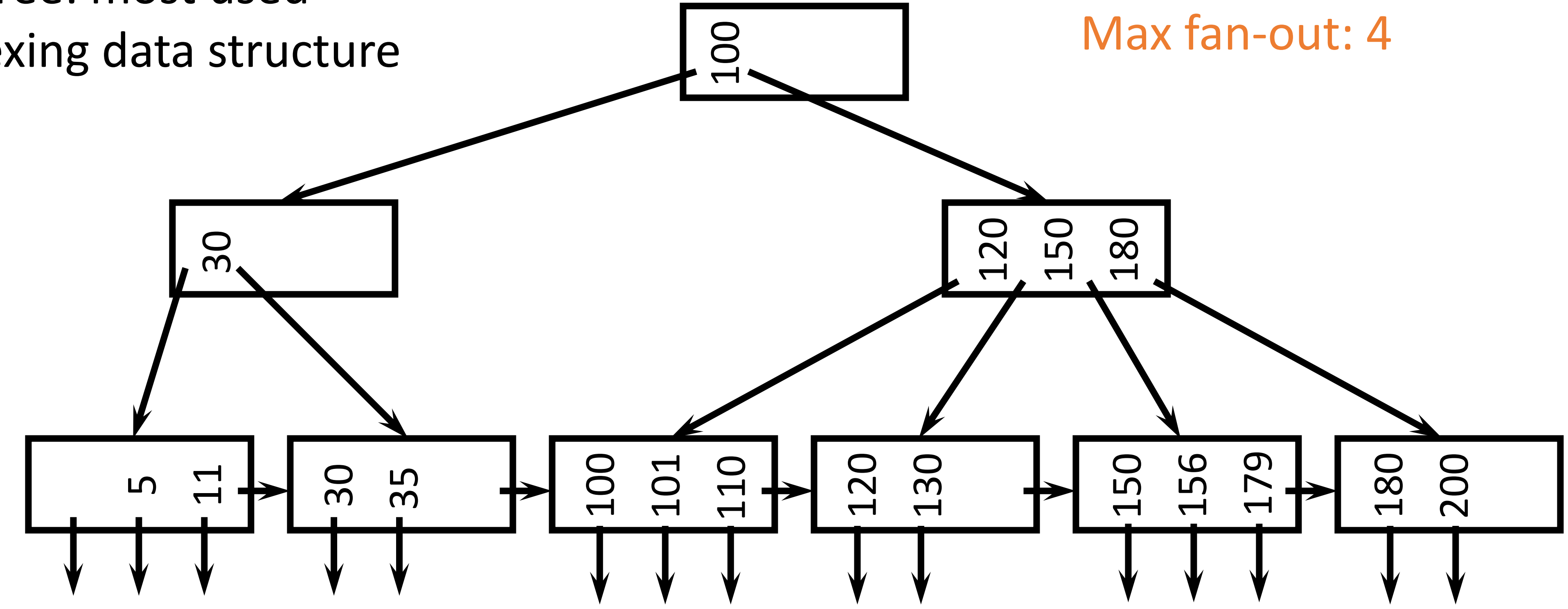
- Aggregates
  - Select AVG(salary) from T1 where age > 20
  - AVG, MIN, MAX, COUNT, SUM
- Join
  - Select * from T1 JOIN T2 ON T1. <column_name> = T2.<column_name>

Indexes

data structures that help find records faster by pointing to loc. where a record is stores

B+ tree: most used indexing data structure

Max fan-out: 4



Query processing

- 3 most common ways: scan based, index based, sort based.

- Example query: Select *name* from T1 where *age < 40* and *age > 20*

- Scan: Assumes no indexes. Linearly scan T1 and check condition on each row. If true, add to output

- Index: Assumes index on *age*. Use the index to fetch all primary keys (pk) satisfying the condition. Then just query those pks from table

- Sort: Assumes data is stored in a sorted order on *age*. Scan only the relevant blocks (stop when conditions are satisfied)

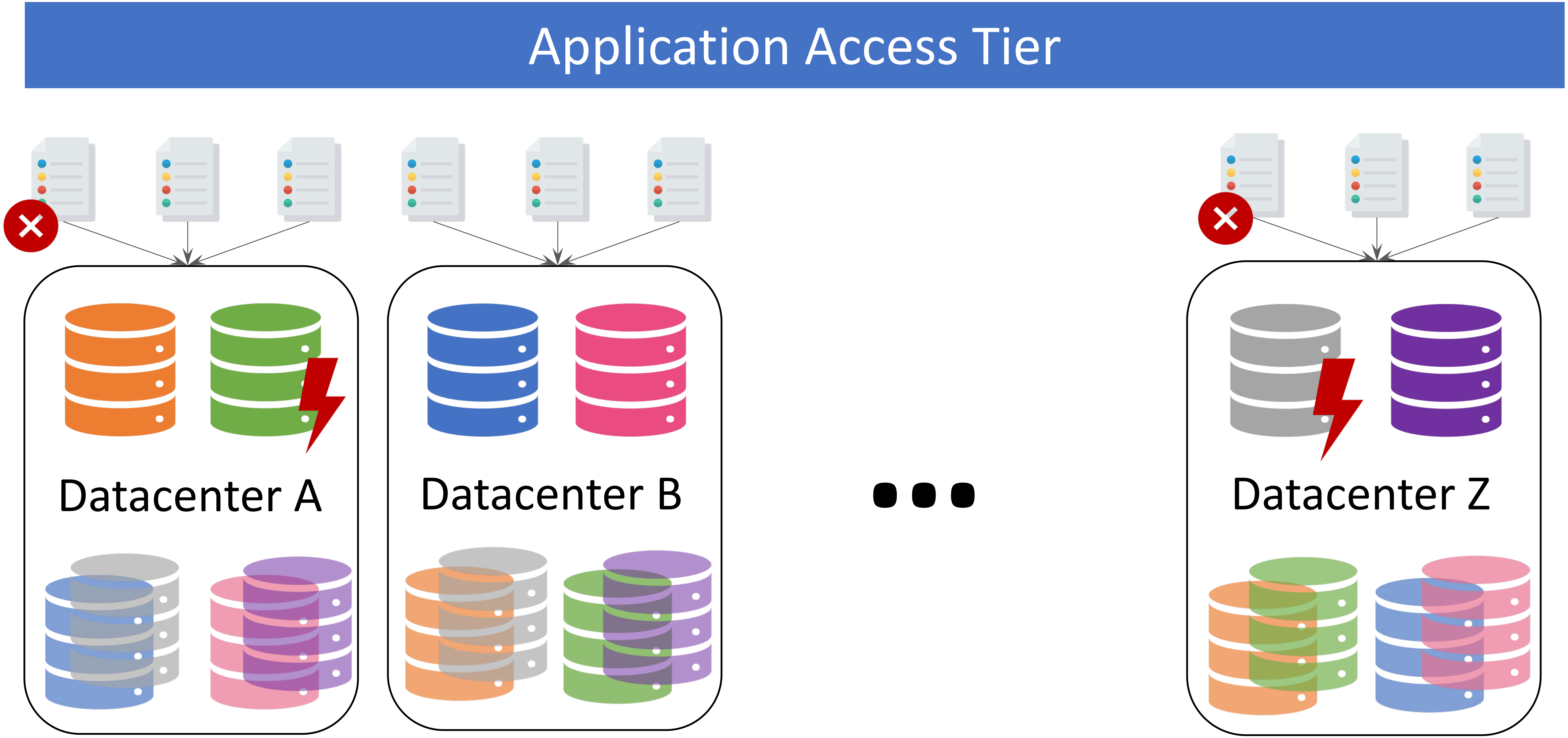
Transactions

- Encapsulates a number of operations – such as select, update, insert – in a single logical unit

- Database systems must ensure ACID
  - Atomicity: TX’s are either completely done or not done at all
  - Consistency: TX’s should leave the database in a consistent state
  - Isolation: TX’s must behave as if they are executed in isolation
  - Durability: Effects of committed TX’s are resilient against failures

Scalability and fault tolerance

By sharding and replicating the data



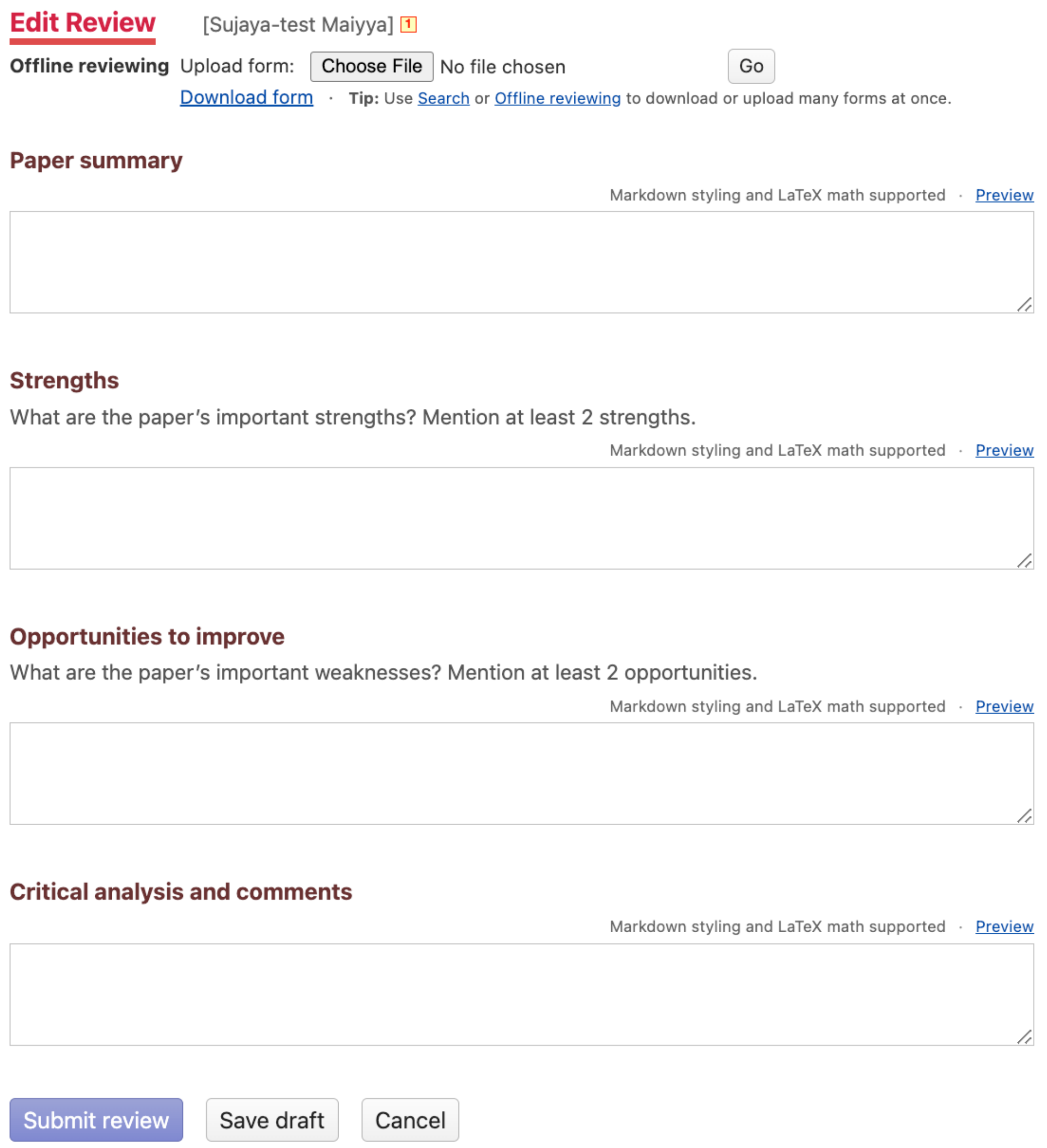
Conclusion

We looked at some basics of cryptography and database concepts

One or more of these techniques will be employed in each paper we will study

HotCRP

- You will receive an invitation to be a Program Committee (PC) member
- Create an account (if you don't already have it) using your Waterloo email id
- Can view papers
- Can submit reviews →



Edit Review [Sujaya-test Maiyya] 1

Offline reviewing Upload form: Choose File No file chosen Go

Download form · **Tip:** Use Search or Offline reviewing to download or upload many forms at once.

Paper summary

Markdown styling and LaTeX math supported · Preview

Strengths

What are the paper's important strengths? Mention at least 2 strengths.

Markdown styling and LaTeX math supported · Preview

Opportunities to improve

What are the paper's important weaknesses? Mention at least 2 opportunities.

Markdown styling and LaTeX math supported · Preview

Critical analysis and comments

Markdown styling and LaTeX math supported · Preview

Submit review | Save draft | Cancel

Questions?