CS848
Background: crypto and db basics

Sujaya Maiyya
One-time pad

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

<table>
<thead>
<tr>
<th>KeyGen:</th>
<th>Enc($k, m \in {0, 1}^\lambda$):</th>
<th>Dec($k, c \in {0, 1}^\lambda$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k \leftarrow {0, 1}^\lambda$ return $k$</td>
<td>return $k \oplus m$</td>
<td>return $k \oplus c$</td>
</tr>
</tbody>
</table>
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 \( c \)

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

vs.

\[
\text{CTXT}(m): \\
c \leftarrow \{0,1\}^\lambda \quad // \text{C of OTP} \\
\text{return } c
\]

“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 \( c \)

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

vs.

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

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

$$\Pr[A \diamond L_{\text{left}} \Rightarrow true] = \Pr[A \diamond L_{\text{right}} \Rightarrow true].$$
Basics of provable security: proving ‘insecurity’

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

<table>
<thead>
<tr>
<th>$L_{\text{otp-real}}$</th>
<th>$L_{\text{otp-rand}}$</th>
<th>$\equiv$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAVESDROP((m \in {0, 1}^\lambda)):</td>
<td>EAVESDROP((m \in {0, 1}^\lambda)):</td>
<td></td>
</tr>
<tr>
<td>(k \leftarrow {0, 1}^\lambda) // OTP.KeyGen</td>
<td>(c \leftarrow {0, 1}^\lambda) // OTP.C</td>
<td></td>
</tr>
<tr>
<td>return (k \oplus m) // OTP.Enc((k, m))</td>
<td>return (c)</td>
<td></td>
</tr>
</tbody>
</table>

What about??

<table>
<thead>
<tr>
<th>$L_{\Sigma\text{-real}}^\Sigma$</th>
<th>$L_{\Sigma\text{-rand}}^\Sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTXT((m \in {0, 1}^\lambda)):</td>
<td>CTXT((m \in {0, 1}^\lambda)):</td>
</tr>
<tr>
<td>(k \leftarrow {0, 1}^\lambda) // $\Sigma$.KeyGen</td>
<td>(c \leftarrow {0, 1}^\lambda) // $\Sigma$.C</td>
</tr>
<tr>
<td>(c := k &amp; m) // $\Sigma$.Enc</td>
<td>return (c = 0^\lambda)</td>
</tr>
<tr>
<td>return (c)</td>
<td></td>
</tr>
</tbody>
</table>

\(\mathcal{A}:\)

- \(c := \text{CTXT}(0^\lambda)\)
- return \(c = 0^\lambda\)

\(\Pr[\mathcal{A} \diamond L_{\text{ots$\$-real \Rightarrow true}} = 1 \quad \Pr[\mathcal{A} \diamond L_{\text{ots$\$-rand \Rightarrow true}} = 1/2^\lambda\)}
Indistinguishability

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

$$\Pr[\mathcal{A} \circ L_{\text{left}} \Rightarrow 1] \approx \Pr[\mathcal{A} \circ L_{\text{right}} \Rightarrow 1] \quad \Rightarrow \quad L_{\text{left}} \approx L_{\text{right}}$$

Advantage or bias of an adversary: $|\Pr[\mathcal{A} \circ L_{\text{left}} \Rightarrow 1] − \Pr[\mathcal{A} \circ L_{\text{right}} \Rightarrow 1]|$

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

<table>
<thead>
<tr>
<th>$\mathcal{L}_{\text{prg-real}}^G$</th>
<th>$\mathcal{L}_{\text{prg-rand}}^G$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QUERY()</strong>:</td>
<td><strong>QUERY()</strong>:</td>
</tr>
<tr>
<td>$s \leftarrow {0, 1}^\lambda$</td>
<td>$r \leftarrow {0, 1}^{\lambda + \ell}$</td>
</tr>
<tr>
<td>return $G(s)$</td>
<td>return $r$</td>
</tr>
</tbody>
</table>

The value $\ell$ 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

\[
\begin{align*}
\text{for } x \in \{0, 1\}^{\text{in}}: \\
T[x] & \leftarrow \{0, 1\}^{\text{out}} \\
\text{LOOKUP}(x \in \{0, 1\}^{\text{in}}): \\
& \text{return } T[x]
\end{align*}
\]

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

\[
\begin{align*}
\mathcal{L}_{\text{prf-real}}^F \\
k & \leftarrow \{0, 1\}^{\lambda} \\
\text{LOOKUP}(x \in \{0, 1\}^{\text{in}}): \\
& \text{return } F(k, x)
\end{align*}
\]

\[
\begin{align*}
\mathcal{L}_{\text{prf-rand}}^F \\
T & := \text{empty assoc. array} \\
\text{LOOKUP}(x \in \{0, 1\}^{\text{in}}): \\
& \text{if } T[x] \text{ undefined:} \\
& \quad T[x] \leftarrow \{0, 1\}^{\text{out}} \\
& \quad T[x] \leftarrow \{0, 1\}^{\text{out}} \\
& \text{return } T[x]
\end{align*}
\]
Pseudorandom Permutations (PRPs) or block ciphers

- Let $K$ be the keyspace, $X$ the message or input space and $Y$ the output space.
- A PRF, $F$:
  \[ F : K \times X \to Y \]
- A PRP, $E$:
  \[ E : K \times X \to X \]
- 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 $\Sigma$ be an encryption scheme. We say that $\Sigma$ has security against chosen-plaintext attacks (CPA security) if $L_{\text{cpa-L}}^{\Sigma} \approx L_{\text{cpa-R}}^{\Sigma}$, where:

<table>
<thead>
<tr>
<th>$L_{\text{cpa-L}}^{\Sigma}$</th>
<th>$L_{\text{cpa-R}}^{\Sigma}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k \leftarrow \Sigma.\text{KeyGen}$</td>
<td>$k \leftarrow \Sigma.\text{KeyGen}$</td>
</tr>
<tr>
<td>$\text{EAVESDROPOPE}(m_L, m_R \in \Sigma.\mathcal{M})$:</td>
<td>$\text{EAVESDROPOPE}(m_L, m_R \in \Sigma.\mathcal{M})$:</td>
</tr>
<tr>
<td>$c := \Sigma.\text{Enc}(k, m_L)$</td>
<td>$c := \Sigma.\text{Enc}(k, m_R)$</td>
</tr>
<tr>
<td>return $c$</td>
<td>return $c$</td>
</tr>
</tbody>
</table>

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 $\text{in} = \lambda$. Define the following encryption scheme based on $F$:

**KeyGen:**
\[
K = \{0, 1\}^\lambda \\
M = \{0, 1\}^{\text{out}} \\
C = \{0, 1\}^\lambda \times \{0, 1\}^{\text{out}}
\]

**Enc($k$, $m$):**
\[
\begin{align*}
& r \leftarrow \{0, 1\}^\lambda \\
& x := F(k, r) \oplus m \\
& \text{return } (r, x)
\end{align*}
\]

**Dec($k$, ($r$, $x$)):**
\[
\begin{align*}
& m := F(k, r) \oplus x \\
& \text{return } m
\end{align*}
\]

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

\[
L^\Sigma_{pk-cpa-L}
\]

$(pk, sk) \leftarrow \Sigma.KeyGen$

GETPK():
return $pk$

CHALLENGE($m_L, m_R \in \Sigma.M$):
return $\Sigma.Enc(pk, m_L)$

\[
L^\Sigma_{pk-cpa-R}
\]

$(pk, sk) \leftarrow \Sigma.KeyGen$

GETPK():
return $pk$

CHALLENGE($m_L, m_R \in \Sigma.M$):
return $\Sigma.Enc(pk, m_R)$
Example – ElGamal encryption

Let \( m \) be the message we want to encrypt

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

Encryption procedure:
\[
E(p, m):
\begin{align*}
&\text{Pick a random number } r \\
&\text{Return } (g^r, m \cdot p^r)
\end{align*}
\]

Decryption procedure:
\[
D(sk, E(p,m)) = B (A^s)^{-1}
\]
\[
= D(s, (g^r, m \cdot p^r))
\]
\[
= (m \cdot p^r) \cdot ((g^r)^s)^{-1}
\]
\[
= (m \cdot (g^s)^r \cdot (g^r)^{-1})
\]
\[
= m
\]
Homomorphic encryption

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

ElGamal encryption is partially homomorphic – supports homomorphic multiplication

\[ E(m_1) \times E(m_2) = (g^{r_1}, m_1 \times p^{r_1}) \times (g^{r_2}, m_2 \times p^{r_2}) \]

\[ = (g^{r_1 + r_2}, (m_1 \times m_2) \times p^{r_1 + r_2}) \]

\[ = E(m_1 \times m_2) \]
Homomorphic encryption (additive)

Modified El Gamal encryption procedure:

\( E(p, m): \)

- Pick a random number \( r \)
- Return \( (g^r, g^m \cdot p^r) \)

This version supports homomorphic additions

\[
E(m1) \cdot E(m2) = (g^{r1}, g^{m1} \cdot p^{r1}) \cdot (g^{r2}, g^{m2} \cdot p^{r2})
\]

\[
= (g^{r1 + r2}, g^{m1 + m2} \cdot p^{r1 + r2})
\]

\[
= E(m1 + m2)
\]
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 pk's 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

Application Access Tier

Datacenter A
Datacenter B
Datacenter Z
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
Questions?