463.4
Crypto Constructs

Computer Security II
CS463/ECE424
University of Illinois
Outline

• Crypto Constructs
  – Homomorphic encryption
  – Searchable encryption
  – Oblivious RAM
  – Private Set Intersection

• Discussion
Symmetric key cryptography
- Same key is used to encrypt and decrypt
- Block ciphers, stream ciphers, modes of operations

Pseudorandom objects
- Collision-resistant hash functions
- Pseudorandom generators (PRGs)

Public key cryptography
- Public key for encryption, private key for decryption
- E.g., RSA
Background: Threat Model

- Ciphertext-only attacks
- Known-plaintext attacks
- Chosen-plaintext attacks
- Chosen-ciphertext attacks

Indistinguishability under Chosen Plaintext Attack (IND-CPA)

- Adversary can’t distinguish pairs of ciphertexts with respect to their plaintexts
- Nondeterministic encryption scheme ($E_K(m)$ is really $E_K(m, r)$ for some random $r$)
463.4.1

Crypto Constructs: Homomorphic Encryption
What if we could...

1. Encrypt data
2. Send it to the cloud
3. Ask the cloud to perform operations
   – Compute, search, sort
• Keeping data encrypted throughout the operation

\[ E(x), f \rightarrow f^*(E(x)) = E(f(x)) \]
Privacy Homomorphisms

• [RAD78] Originally idea introduced by Rivest, Adleman, and Dertouzos
• Proposed several privacy homomorphisms, but none of them were secure against chosen-plaintext attacks

Privacy homomorphism: Operators (□, ○) such that \( E(x) \circ E(y) = E(x \square y) \)
Homomorphic Encryption

• Fully Homomorphic Encryption (FHE)
  – Two operations: e.g., addition and multiplication
  – $E(x \cdot (y + z)) = E(x) \circ (E(y) \square E(z))$
  – [Gentry09] First scheme
  – Not efficient

• Partially Homomorphic Encryption (PHE)
  – Only one operation: e.g., only multiplication
  – $E(x \cdot y) = E(x) \circ E(y)$
  – Many public-key cryptosystems are partially homomorphic, e.g., RSA - Fairly efficient
Plain RSA

Setup:
p and q large primes, \( N = pq \), \( \varphi = (p-1)(q-1) \),
Take \( e \) coprime with \( \varphi \), and calculate \( d = e^{-1} \mod \varphi \),
\( K' = (N, d) \) is the private key

- Alice
- **K** ← (\( N, e \))
- Bob
- \( c ← m^e \mod N \)
- Decryption
  \( m = c^d \mod N \)

[RSA76]
RSA

Setup:
p and q large primes, \( N = pq \), \( \varphi = (p-1)(q-1) \),
Take \( e \) coprime with \( \varphi \),
d = \( e^{-1} \mod \varphi \), \( K' = (N, d) \)

Alice

Bob

Messages \( m_1, m_2 \)

\( c_1 \leftarrow m_1^e \mod N \), \( c_2 \leftarrow m_2^e \mod N \)

\( c \leftarrow c_1 \cdot c_2 \mod N \)

\( m_1 \cdot m_2 \leftarrow c^d \mod N \)

Plain RSA is a privacy homomorphism with respect to multiplication: \( E_K(xy) = E_K(x) \cdot E_K(y) \).
But it does not provide ciphertext indistinguishability (i.e., encryption is not randomized)
Applications of PHE

• e-Voting
  – Protect the anonymity of the voters

• Digital cash
  – Ensure anonymity

• Mix networks
  – Re-randomization of ciphertexts

• Private Matching / Private Set Intersection
  – Search for members of a terrorism watch list in an air flight passenger list
Additive Homomorphic Encryption

• Addition
  – $E_K(m_1) \circ E_K(m_2) = E_K(m_1 + m_2)$

• Multiplication (by a constant c)
  – $E_K(m)^c = E_K(m) \circ \ldots \circ E_K(m) = E_K(c \cdot m)$

• Schemes in practice are IND-CPA secure; i.e., provide randomized encryptions
  – $E_K(m)$ is really $E_K(m, r)$, for some random $r$
  – Re-randomization: $E_K(m) \circ E_K(0) = E_K(m)$
1. Trusted
   – Ask the cloud to do computation / search in plaintext

2. Honest-but-curious (aka semi-honest)
   – Cloud can try to learn more information; perform statistical inferences, or try to break the crypto
   – Cloud cannot deviate from protocol
   – Captures threats by curious sys admins

3. Malicious
   – Cloud can deviate arbitrarily from protocol
Other Crypto Constructs

• Yao’s garbled circuits
  – Client prepares a garbled (i.e., encrypted) circuits
  – Server evaluates the circuit

• Secret sharing
  – Secret is split among multiple parties
  – Collaboration of a majority of the parties is required to recover the secret and/or perform a computation
463.4.2
Crypto Constructs:
Searchable Encryption and ORAM
Client wants to search for documents which contain a specific keyword

Can the search be outsourced to a server without revealing the contents of the documents or the search keyword?

– Client encrypts the documents, sends them to server
– Client asks the server to return the (encrypted) documents containing a particular keyword
Searchable Encryption

**Initialization**

- **Client**
- **Server**

**Search**

- **Client**
  - Keyword $w$
- **Server**
  - Encrypted keyword $w$
Searchable Encryption

- Naive solution
  - Encrypt keywords (with a deterministic scheme)

<table>
<thead>
<tr>
<th>Encrypted Keyword</th>
<th>Document IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E(w_1)$</td>
<td>1, 7, 16</td>
</tr>
<tr>
<td>$E(w_2)$</td>
<td>3, 5</td>
</tr>
<tr>
<td>$E(w_3)$</td>
<td>7</td>
</tr>
<tr>
<td>$E(w_4)$</td>
<td>13, 11, 5, 2, 1</td>
</tr>
</tbody>
</table>

Client: Search for keyword $w_2$
Searchable Encryption

• Possible guarantees: the server learns only
  1. Keyword access pattern (i.e., last time this keyword was searched)
  2. Document access pattern (i.e., documents that are accessed for each keyword search)

• Reveals more in practice due to updates (e.g., add a document, delete a document)
Access pattern leaks

• With auxiliary information:
  – Multi-user systems: correlate queries
  – Information about user: e.g., EMR of a patient is accessed by an oncologist

• Identify 80% of search queries on encrypted emails using access pattern alone
Software Protection and ORAM

• [GO96] Oblivious RAM - Originally proposed for software protection by Goldreich and Ostrovsky

• Approach to software protection:
  – Tamperproof CPU and encrypted program
  – Decryption key embedded in ROM inside CPU
  – For each instruction: fetch, decrypt, execute

• RAM content can be encrypted, but program execution reveals the memory addresses accessed

• Related but not the same: e.g., TPM, and Intel SGX
Oblivious RAM

- Idea: access a RAM with \( N \) memory cells in a way that is *independent* of the program / input
- Oblivious if for any two inputs (request sequences) of the same length, the access patterns are *equivalent* (i.e., indistinguishable)

- Trivial solution: access every memory cell for each request
- \([\text{GO96}]\) First solution – \( O(N^{\frac{1}{2}}) \) overhead (i.e., average number of accesses for each request)
- Current best - \( O(\log N) \) overhead – 20-40X in practice
Oblivious RAM

• How to hide the access pattern & frequency?
• Intuition:
  – Use a non-deterministic encryption scheme
    • Every time a block is re-encrypted, the ciphertext is different, even for the same plaintext
  – Move blocks around & reshuffle periodically (i.e., permuting blocks randomly)
  – Some schemes use local caching (e.g., to hide access frequency)
Oblivious RAM: Square Root Algorithm

1. For each of $N^{\frac{1}{2}}$ requests:
   - Look for block in the shelter
   - If found, access the next dummy index
   - If not found, get word from permuted memory and put it in the shelter

2. After $N^{\frac{1}{2}}$ requests:
   - Reshuffle the permuted memory, obliviously updating it with the values in the shelter
463.6
Crypto Constructs:
Private Set Intersection
• How can companies share cybersecurity incidents with each other without revealing unnecessary information?
  – Each group will randomly select 10 incidents
  – Each group will run a Private Set Intersection (PSI) protocol to see if other groups have experienced the same incidents
  – You will need to implement a (small) part of the PSI protocol
  – What can you learn in the honest-but-curious setting? What about the malicious setting?
Private Set Intersection Cardinality (PSI-CA)

Server

\[ S = \{ s_1, \ldots, s_m \} \]

Client

\[ C = \{ c_1, \ldots, c_n \} \]

Private Set Intersection Cardinality (PSI-CA)

\[ |S \cap C| \]
Private Set Intersection

- Client has a set $C$ of $n$ items
- Server has a set $S$ of $m$ items
- We want to compute $C \cap S$ (or $|C \cap S|$) without revealing anything more about $C$ and $S$

- Approach:
  1. Express $C$ as a polynomial $P(X)$
  2. Server evaluates $P(X)$ at each $s \in S$ using additive homomorphic encryption
Private Set Intersection

\[ C = \{c_1, \ldots, c_n\} \]
\[ P(X) = \prod_{i=1}^{n} (X - c_i) = \sum_{j=0}^{n} a_j X^j \]

For each \( s_j \in S \):
- Pick a random \( r_j \)
- Homomorphically evaluate \( P(s_j) \)
- \( E_K(r_j \, P(s_j) + s_j) \)

\[ E_K(r_1 P(s_1) + s_1), \ldots, E_K(r_m P(s_m) + s_m) \]

- If \( c_i = s_j \), then \( E_K(r_j \, P(s_j) + s_j) = E_K(s_j) \)
- Otherwise \( E_K(r_j \, P(s_j) + s_j) = E_K(r) \), for some random \( r \)
Additive Homomorphic Encryption

• Addition
  – $E_K(m_1) \circ E_K(m_2) = E_K(m_1 + m_2)$

• Multiplication (by a constant $c$)
  – $E_K(m)^c = E_K(m) \circ \ldots \circ E_K(m) = E_K(c \cdot m)$

• Schemes in practice are IND-CPA secure; i.e., provide randomized encryptions
  – $E_K(m)$ is really $E_K(m, r)$, for some random $r$
  – Re-randomization: $E_K(m) \circ E_K(0) = E_K(m)$
• How does the server compute $E_K(r_i P(s_i) + s_i)$?
  – Pick random $r_i$
  – Evaluate $P(s_i)$ using $E_K(a_0), \ldots, E_K(a_n)$ from client
    • Recall that $P(X) = \prod_{i=1}^n (X - c_i) = \sum_{j=0}^n a_j X^j$
    • For $j=0,\ldots,n$: compute $s_i^j$, then homomorphically compute
      $E_K(a_j)^{s_i^j} = E_K(a_j s_i^j)$
    • Homomorphically sum the terms by computing:
      $\prod_{j=0}^n E_K(a_j s_i^j) = E_K[\sum_{j=0}^n a_j s_i^j] = E_K[P(s_i)]$
    • $E_K[P(s_i)]^{r_i} \circ E_K[s_i] = E_K(r_i P(s_i) + s_i)$
References


Discussion Questions

• Why not just trust the cloud provider?
• Are there alternative architectures for searchable encryption?
  – Keep the index on the client?
  – Use two cloud providers?
• What other problems could be solved using Private Set Intersection?