Crypto Constructs

CS463/ECE424

University of Illinois

Crypto Constructs

Homomorphic Encryption
Private Set Intersection
Searchable Encryption

Recap: Basic Crypto Concepts

- Symmetric key cryptography
  - Same key is used to encrypt and decrypt
  - Block ciphers, stream ciphers

- Public key cryptography
  - Public key for encryption, private key for decryption
  - E.g., RSA

- Collision-resistant hash functions

Background: Threat Model (1)

Attack Goal: get target plaintext

- Ciphertext-only attacks
  - Attacker only has access to the ciphertext
  - Most realistic
- Known-plaintext attacks
  - Attacker has access to a set of (plaintext, ciphertext) pairs
- Chosen-plaintext attacks
  - Attacker can pick arbitrary plaintext and obtain corresponding ciphertext
- Chosen-ciphertext attacks
  - Attacker can pick arbitrary ciphertext and obtain corresponding plaintext
  - Strongest attacker

Background: Threat Model (2)

Indistinguishability under Chosen Plaintext Attack (IND-CPA)

- Adversary can't distinguish pairs of ciphertexts with respect to their plaintexts
  - I.e., Give the attacker C1 = Enc(P1), C2 = Enc(P2) and P1, P2, and ask the attacker to create the mapping b/w Pi and Cj
- Requires nondeterministic encryption scheme ($E_K(m)$ is really $E_K(m, r)$ for some random r)

Why do we need randomized encryption? (1)

IND-CPA Game:

- First the challenger generates an encryption keypair and sends the public key *pk* to the adversary. (It keeps the secret key.)
- Next, the adversary selects a pair of messages M_0, M_1 (of equal length) and sends them to the challenger.
- The challenger picks a random bit $b \in \{0,1\}$ and encrypts one of the two messages as $C^* \leftarrow Enc(M_b, pk)$. It sends back C^* to the adversary.
- Finally, the adversary outputs a guess b'. We say the adversary "wins" if it guesses correctly: that is, if $b' = b$.

https://blog.cryptographyengineering.com/why-ind-cpa-implies-randomized-encryption/

Why do we need randomized encryption? (2)

- First the challenger generates an encryption keypair and sends the public key *pk* to the adversary. (It keeps the secret key.)
- Next, the adversary selects a pair of messages *M_0, M_1* (of equal length) and sends them to the challenger.
- The challenger picks a random bit $b \in \{0,1\}$ and encrypts one of the two messages as $C^* \leftarrow Enc(M\_b, pk)$. It sends back C^* to the adversary.
- Finally, the adversary outputs a guess b'. We say the adversary “wins” if it guesses correctly: that is, if $b' = b$.

If encryption is not randomized..

- The adversary picks two messages *M_0, M_1* and then encrypts both of them using the public key.
- When the adversary receives the ciphertext C^*, he just compares that ciphertext to the two he generated himself.
- Voila, the adversary can always figure out which message was encrypted i.e., encryption fails!!

https://blog.cryptographyengineering.com/why-ind-cpa-implies-randomized-encryption/

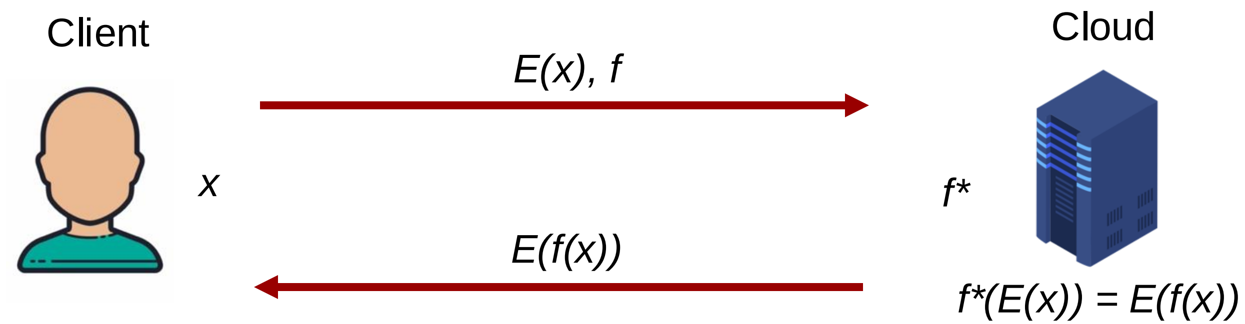
Homomorphic Encryption

What if we could…

1. Encrypt data
2. Send it to the cloud
3. Ask the cloud to perform operations
   – Compute, search, sort

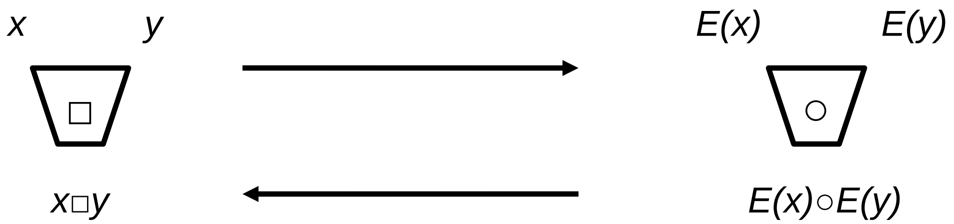
Keeping data encrypted throughout the operation!

Who would be interested in such technique?



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
  - Mostly because the encryption scheme is not randomized



Privacy homomorphism: Operators (□, ○) such that *E(x)○E(y) = E(x□y)*

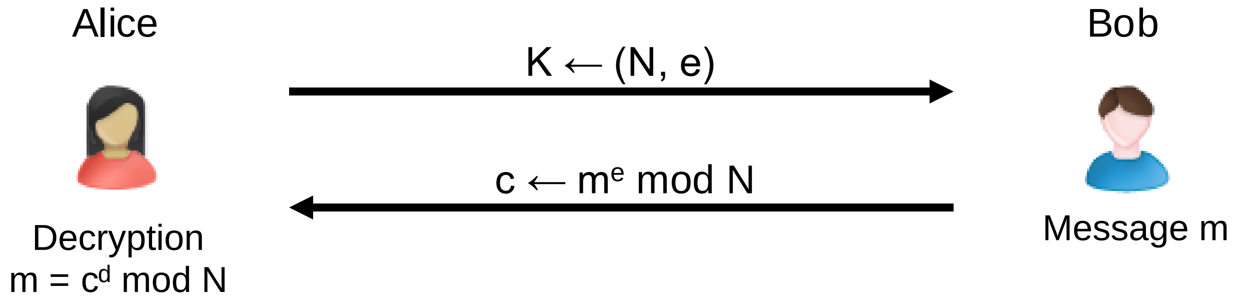
Homomorphic Encryption

- Fully Homomorphic Encryption (FHE)
  - Two operations: e.g., addition and multiplication
  - $E(x\,(y + z)) = E(x)\ \Delta\ (E(y) \circ E(z))$
  - [Gentry09] First scheme
  - Not efficient
- Partially Homomorphic Encryption (PHE)
  - Only one operation: e.g., only multiplication
  - $E(x\,y) = E(x)\ \Delta\ E(y)$
  - Many public-key cryptosystems are partially homomorphic
  - e.g., RSA - Fairly efficient

Plain RSA

Setup:

- p and q large primes, $N = pq$, $z = (p-1)(q-1)$,
- Take e co-prime with z, and calculate $d = e^{-1} \bmod z$,
- $K' = (N, d)$ is the private key
- $K = (N, e)$ is the public key



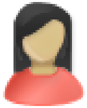
[RSA76]

RSA

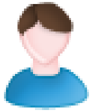
Setup:

- p and q large primes, $N = pq$, $z = (p-1)(q-1)$,
- Take e coprime with z, $d = e^{-1} \bmod z$
- $K' = (N, d)$

Alice



Bob



$K \leftarrow (N, e)$ →

Messages m_1, m_2

$c_1 \leftarrow m_1^e \bmod N$, $c_2 \leftarrow m_2^e \bmod N$ →

← $c \leftarrow c_1 \cdot c_2 \bmod N$

$m_1 \cdot m_2 \leftarrow c^d \bmod N$

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)

Additive Homomorphic Encryption

- Addition
  - $E_K(m_1) \circ E_K(m_2) = E_K(m_1 + m_2)$
- Multiplication (by a constant c)
  - $E_K(m)^c = E_K(m) \circ \ldots \circ E_K(m) = E_K(c \cdot m)$

"Bonus" operation derived from addition

Applications of PHE

- e-Voting
  - Calculate the total the votes without seeing plaintext votes
  - Protect the anonymity of the voters

- Digital cash
  - Ensure anonymity over financial transactions

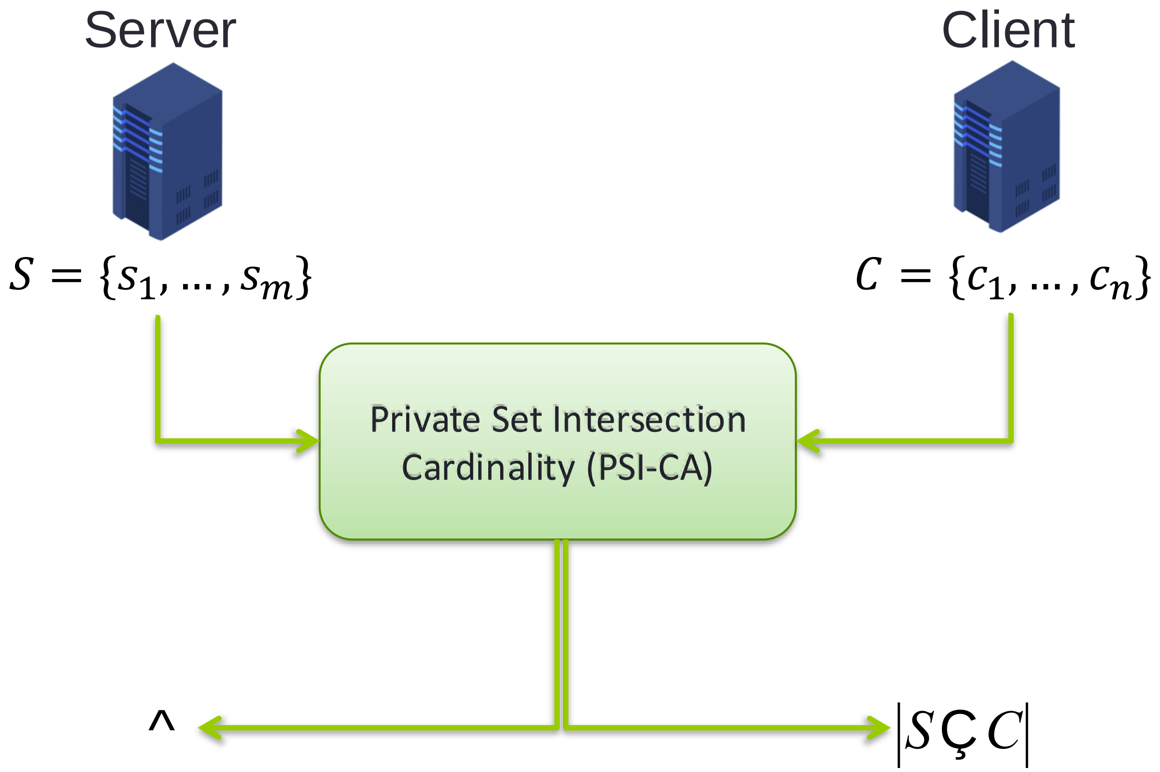
- Private Matching / Private Set Intersection
  - Search for members of a watch list in an air flight passenger list

Threat Model (think about cloud computing)

1. **Model in the status quo:** Trusted
   – Ask the cloud to do computation / search in plaintext
2. Honest-but-curious (aka semi-honest)
   – Cloud cannot deviate from the protocol (i.e., honest)
   – Cloud can try to learn more information; perform statistical inferences, or try to break the crypto (i.e., curious)
   – Captures threats by curious system admins
3. Malicious
   – Cloud can deviate arbitrarily from protocol

Private Set Intersection

Private Set Intersection Cardinality (PSI-CA)



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

Client

Public key: K
Secret key: K'

$C = \{c_1, \ldots, c_n\}$

$$P(X) = \prod_{i=1}^{n} (X - c_i) = \sum_{l=0}^{n} a_l X^l$$

$E_K(a_0), \cdots, E_K(a_n)$ →

Server

Public key: K

$S = \{s_1, \ldots, s_m\}$

For each $s_j \in S$:

- Pick a random r_j
- **Homomorphically evaluate** $P(s_j)$
- $E_K(r_j\, P(s_j) + s_j)$

Recall: Additive Homomorphic Encryption

- Addition
  - $E_K(m_1) \circ E_K(m_2) = E_K(m_1 + m_2)$
- Multiplication (by a constant c)
  - $E_K(m)^c = E_K(m) \circ \ldots \circ E_K(m) = E_K(c \cdot m)$

Private Set Intersection

Client

$C = \{c_1, \dots, c_n\}$

Server

$S = \{s_1, \dots, s_m\}$

How does the server compute $E_K(r_j P(s_j) + s_j)$?

- Recall: For each s_j, pick a random r_j
- Evaluate $P(s_j)$ using $E_K(a_0), \dots, E_K(a_n)$ received from client
  - Recall that $P(X) = \prod_{i=1}^{n}(X - c_i) = \sum_{l=0}^{n} a_l X^l$
  - For $l = 0, \dots, n$:
    - compute ${s_j}^l$
    - then homomorphically compute $E_K(a_l)^{{s_j}^l} = E_K(a_l {s_j}^l)$ (multiplication by a constant)
  - Homomorphically sum the terms by computing: $\prod_{l=0}^{n} E_K(a_l {s_j}^l) = E_K[\sum_{l=0}^{n} a_l {s_j}^l] = E_K[P(s_j)]$
  - $E_K[P(s_j)]^{r_j} \circ E_K[s_j] = E_K(r_j P(s_j) + s_j)$

Normal addition

Homomorphic addition

Private Set Intersection

Client

Public key: K

Secret key: K'

Server

Public key: K

$$C = \{c_1, \dots, c_n\}$$

$$P(X) = \prod_{i=1}^{n} (X - c_i) = \sum_{l=0}^{n} a_l X^l$$

$$S = \{s_1, \dots, s_m\}$$

For each $s_j \in S$:

- Pick a random r_j
- **Homomorphically evaluate** $P(s_j)$
- $E_K(r_j\ P(s_j) + s_j)$

$E_K(a_0), \cdots, E_K(a_n)$ → (Client to Server)

$E_K(r_1 P(s_1) + s_1), \cdots, E_K(r_m P(s_m) + s_m)$ ← (Server to Client)

- Client: perform intersection on the encrypted values:
  - If $c_i = s_j$, then $P(s_j) = 0$, and thus $E_K(r_j\ P(s_j) + s_j) = E_K(s_j) = E_K(c_i)$
  - Otherwise $E_K(r_j\ P(s_j) + s_j) = E_K(r)$, for some random r

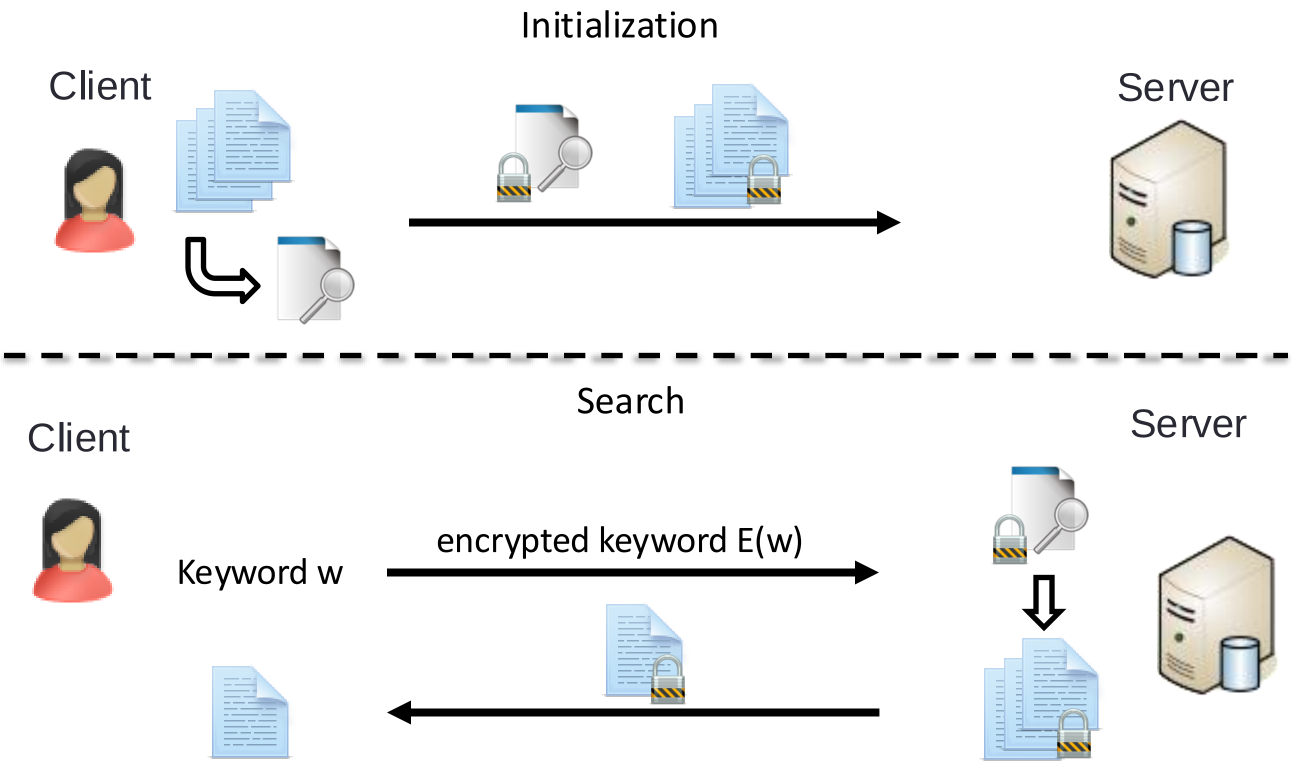
Searchable Encryption

Searchable Encryption

- 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

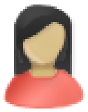


Searchable Encryption

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

Server

Client



Search for keyword w_2

$E(w_2)$ →

Encrypted Keyword	Document IDs
$E(w_1)$	1, 7, 16
$E(w_2)$	3, 5
$E(w_3)$	7
$E(w_4)$	13, 11, 5, 2, 1

Encrypted Index

Cons: Index list will be HUGE!

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 users who send the query: e.g., EMR of a patient is accessed by an oncologist
- Identify 80% of search queries on encrypted emails using access pattern alone
  - E.g., based on word distribution in emails

[IKK12] Islam, M., Mehmet Kuzu, and Murat Kantarcioglu. "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation." NDSS 2012.

[CGPR15] Cash, D., Grubbs, P., Perry, J. and Ristenpart, T. “Leakage-abuse attacks against searchable encryption.” ACM CCS 2015.

How to Make Accesses Oblivious?

“Doesn’t look like anything to me.”

Software Protection and ORAM (Extra Reading)

- [GO96] Oblivious RAM - Originally proposed for software protection by Goldreich and Ostrovsky
- Traditional approach to software protection:
  - Tamperproof CPU and encrypted program
  - Decryption key embedded in ROM inside CPU
  - For each instruction: fetch, decrypt, execute
  - Protect RAM content from the rest of the system
- RAM content can be encrypted, but program execution reveals the memory addresses accessed → motivation for **Oblivious RAM**

[GO96] Goldreich, and Ostrovsky. "Software protection and simulation on oblivious RAMs." Journal of the ACM (JACM) 1996.

References

- [RAD78] Rivest, Adleman, and Dertouzos. "On data banks and privacy homomorphisms." Foundations of secure computation 4.11 (1978).
- [CGPR15] Cash, D., Grubbs, P., Perry, J. and Ristenpart, T. “Leakage-abuse attacks against searchable encryption.” ACM CCS 2015.
- [GO96] Goldreich, and Ostrovsky. "Software protection and simulation on oblivious RAMs." Journal of the ACM (JACM) 1996.
- [FNP04]: Freedman, Nissim, Pinkas. "Efficient private matching and set intersection." EUROCRYPT 2004.

Discussion Questions

- Why not just trust the cloud provider?

- What other problems could be solved using Private Set Intersection?

- Are there alternative architectures for searchable encryption?
  - Keep the index on the client?
  - Use two cloud providers?