463.4 Crypto Constructs

Computer Security II
CS463/ECE424
University of Illinois

Security News of the Day

https://www.zdnet.com/article/apple-slams-the-brakes-on-plans-to-scan-user-iphones-for-child-abuse-content/

Apple slams the brakes on plans to scan user images for child abuse content

Backlash stemming from privacy concerns has delayed the rollout.

By Charlie Osborne for Zero Day | September 6, 2021 -- 08:25 GMT (01:25 PDT) | Topic: Security

"The features Apple announced a month ago, intending to help protect children, would create an infrastructure that is all too easy to redirect to greater surveillance and censorship," the digital rights group says. "These features would create an enormous danger to iPhone users' privacy and security, offering authoritarian governments a new mass surveillance system to spy on citizens."

Associate Professor at the Johns Hopkins Information Security Institute and cryptography expert Matthew Green said the implementation of cryptography to scan for images containing specific hashes could become "a key ingredient in adding surveillance to encrypted messaging systems."

Security News of the Day

https://www.forbes.com/sites/zakdoffman/2021/09/12/is-facebook-spying-on-whatsapp-messages-on-iphone-android-mac-windows-10/?sh=2c7e2a42764f

Sep 12, 2021, 06:30am EDT | 3,701 views

Is Facebook 'Secretly' Spying On Your WhatsApp Messages?

Ars Technica
@arstechnica

WhatsApp "end-to-end encrypted" messages aren't that private after all arstechnica.com/gadgets/2021/0... by @jrssnet

ProPublica
@propublica

Update: We've altered language in our WhatsApp story to clarify that the company examines only messages from threads that have been reported by users as possibly abusive. It does not break end-to-end encryption.

Crypto Constructs

Homomorphic encryption
Private Set Intersection
Searchable encryption
Oblivious RAM

Background: Landscape

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

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

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

Background: Threat Model

Attack Goal: get target plaintext

- 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)

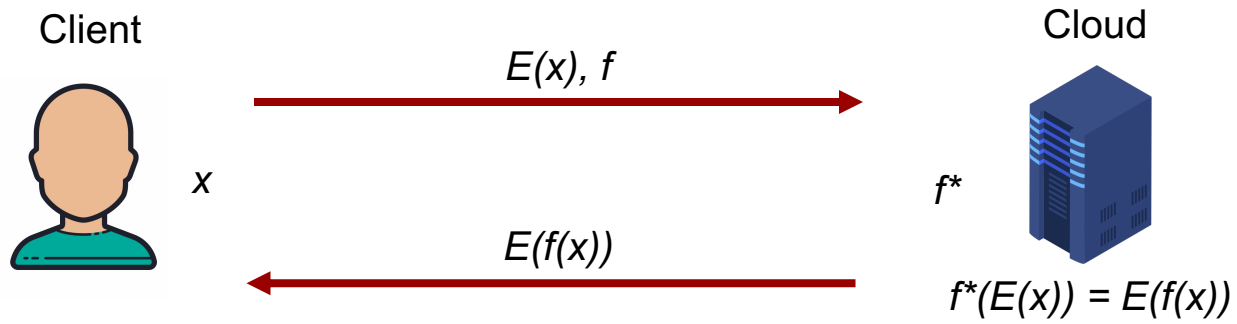
463.4.1
Crypto Constructs: Homomorphic Encryption

What if we could…

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

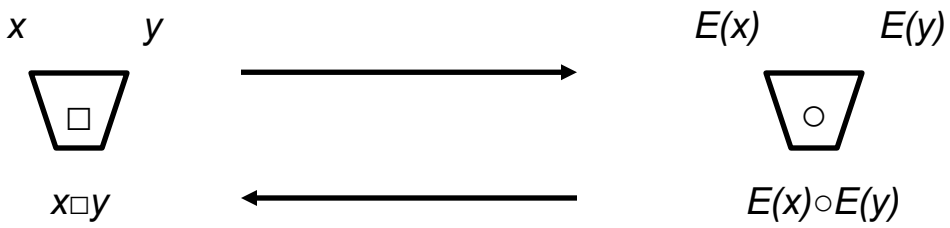
Keeping data encrypted throughout the operation!

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



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

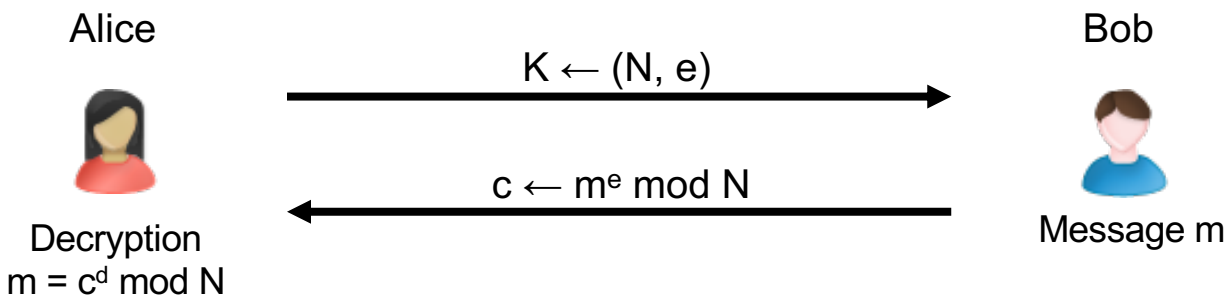
Homomorphic Encryption

- Fully Homomorphic Encryption (FHE)
  - Two operations: e.g., addition and multiplication
  - $E(x\ (y + z)) = E(x)\ \Delta\ (E(y) \circ E(z))$
  - [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, $N = pq$, $\varphi = (p-1)(q-1)$,
- Take e coprime with φ, and calculate $d = e^{-1} \bmod \varphi$,
- $K' = (N, d)$ is the private key
- $K = (N, e)$ is the public key



[RSA76]

RSA

Setup:

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

Alice

Bob

$K \leftarrow (N, e)$ →

Messages m_1, m_2

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

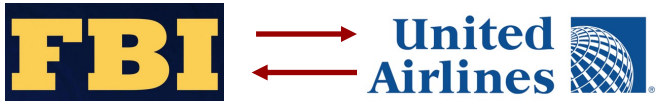
$m_1 \cdot m_2 \leftarrow c^d \bmod N$ ← $c \leftarrow c_1 \cdot c_2 \bmod N$

Plain RSA is a privacy homomorphism with respect to <u>multiplication</u>: $E_K(xy) = E_K(x) \cdot E_K(y)$.

But it does not provide ciphertext indistinguishability (i.e., encryption is not randomized)

Applications of PHE

- e-Voting
  - Calculate the total the votes without seeing plaintext votes
  - Protect the anonymity of the voters
- Digital cash
  - Ensure anonymity over financial transactions
- Mix networks
  - Re-randomization of ciphertexts for anonymity
- 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)
  - $E_K(m)^c = E_K(m) \circ \ldots \circ E_K(m) = E_K(c \cdot m)$

"Bonus" operation derived from addition

- 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: $E_K(m) \circ E_K(0) = E_K(m)$

Threat Model

1. 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

463.4.2 Crypto Constructs: Private Set Intersection

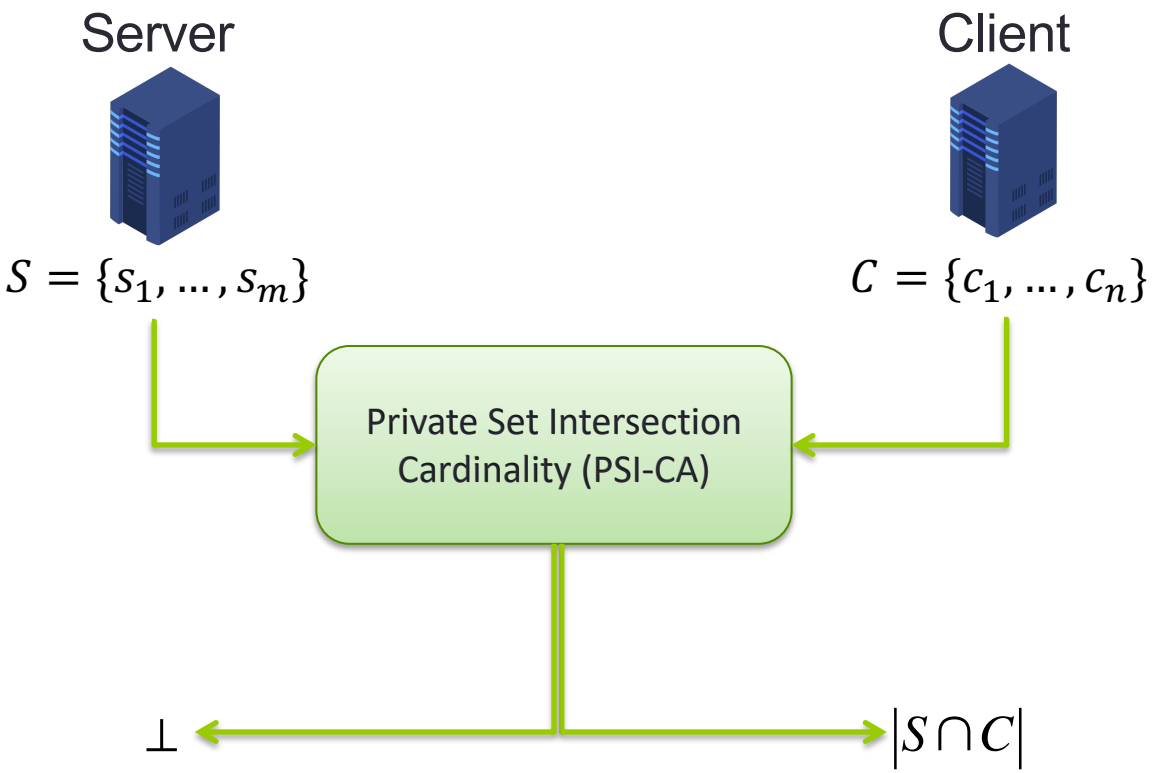
MP2

- How can companies share security incidents information with each other without revealing unnecessary information?
  - Each company will randomly select 10 incidents
  - Each company will run a Private Set Intersection (PSI) protocol to see if other companies have experienced the same incidents
  - You will act as a company and 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)



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'

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

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

Server

Public key: K

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

For each $s_j \in S$:

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

How? Next slides

$$\xrightarrow{E_K(a_0), \cdots, E_K(a_n)}$$

$$\xleftarrow{E_K(r_1 P(s_1) + s_1), \cdots, E_K(r_m P(s_m) + s_m)}$$

- Client: perform intersection on the encrypted values:
  - If $c_i = s_j$, then $P(s_j) = 0$, and thus $\boxed{E_K(r_j P(s_j) + s_j)} = E_K(s_j) = \boxed{E_K(c_i)}$
  - 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: $E_K(m) \circ E_K(0) = E_K(m)$

Private Set Intersection

Client

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

Server

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

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

- 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: $\prod_{l=0}^{n} E_K(a_l {s_j}^l) = E_K[\sum_{l=0}^{n} a_l {s_j}^l] = E_K[P(s_j)]$
  - $E_K[P(s_j)]^{r_j} \circ E_K[s_j] = E_K(r_j P(s_j) + s_j)$

Homomorphic addition

Discussion Questions

- Why not just trust the cloud provider?

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

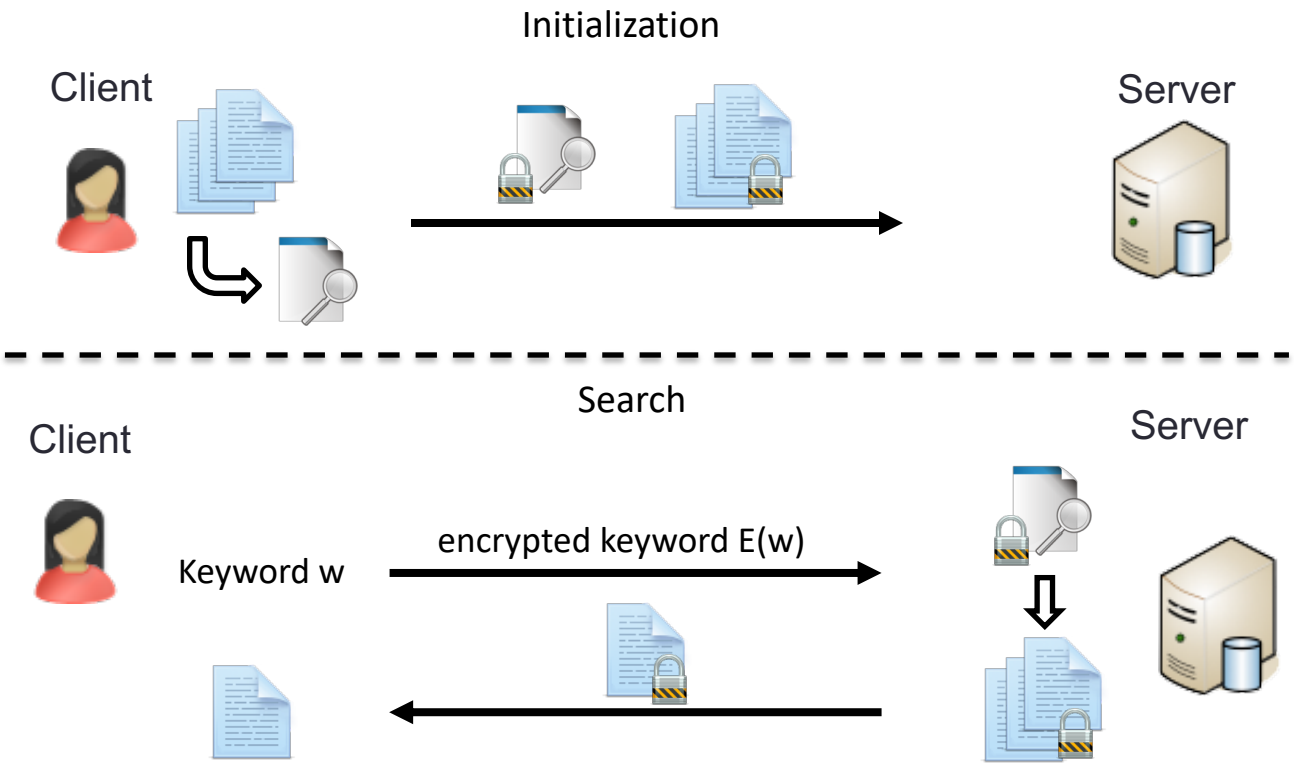
463.4.3 Crypto Constructs: Searchable Encryption and ORAM

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

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?

Software Protection and ORAM

- [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**

Oblivious RAM

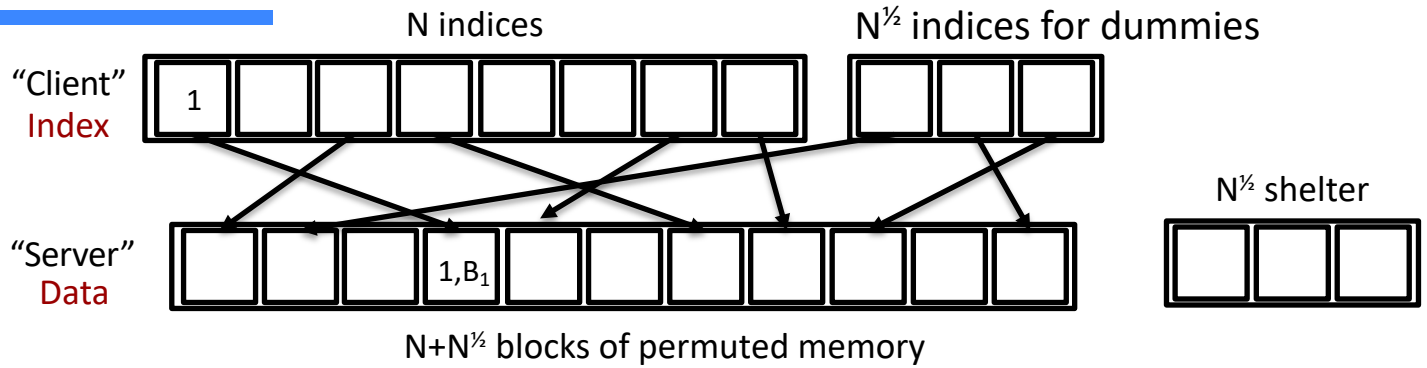
- Idea: access a RAM with N memory cells in a way that is *independent* of the program / input
- Oblivious if for any two inputs (request sequences) of the same length, the access patterns are *equivalent* (i.e., indistinguishable)

- Trivial solution: full redundancy: access every memory cell for each request --- oblivious, but too costly
- [GO96] First solution – $O(N^{1/2})$ 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?
- Intuitions:
  - 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)
  - Use local caching (e.g., to hide access frequency)

Oblivious RAM: Square Root Algorithm



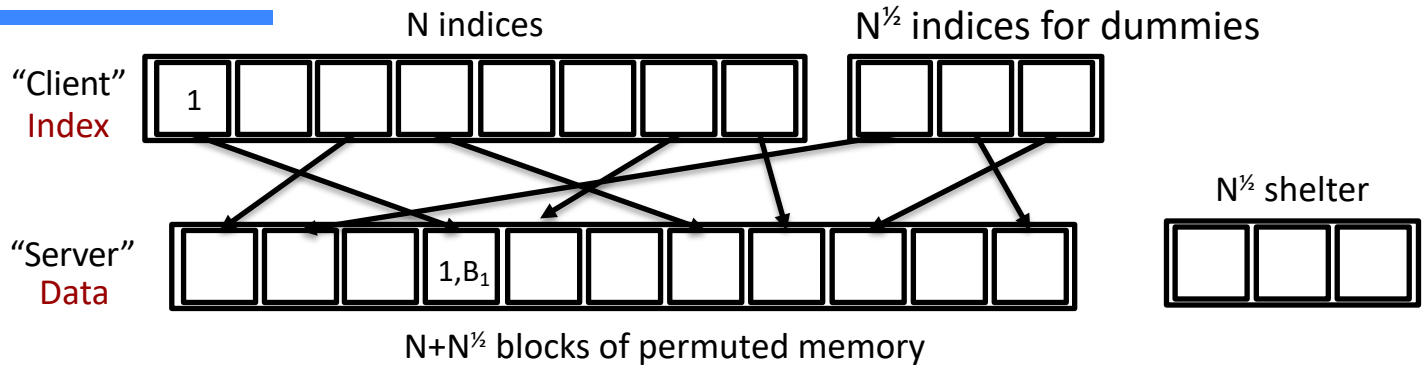
Setups:

- **Client:** has the index which tells you where things are on the server
- **Server:** holds the memory blocks to store your data

Each epoch (you can perform $N^{1/2}$ queries/searches)

- Access one memory cell at most once --- to avoid leaking frequency patterns
- Once a cell is accessed, place the data to the <u>shelter</u>
  - In case we need to access it again in this epoch, access via shelter (caching)

Oblivious RAM: Square Root Algorithm

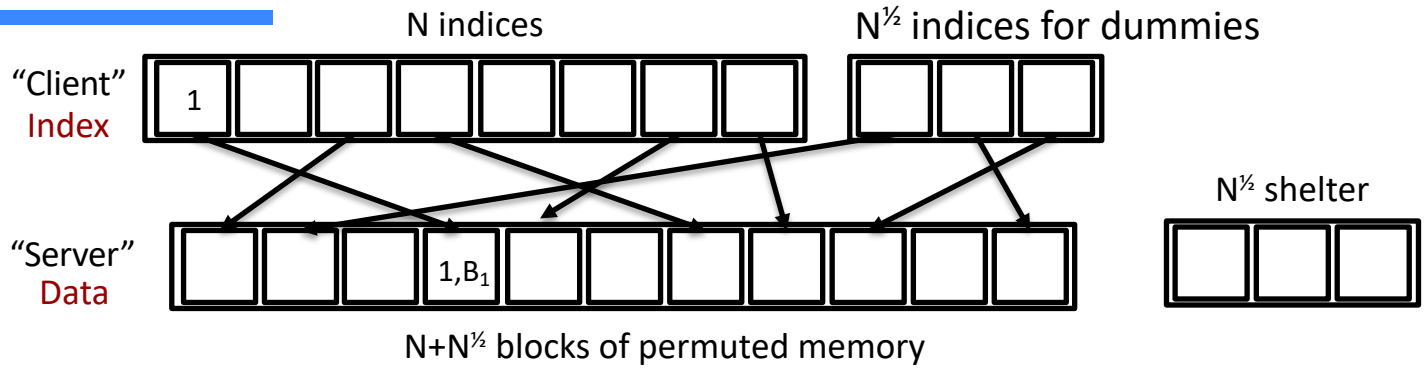


For each request within the epoch (good for N½ requests):

- Always look for the block in the shelter first
  - If found, access the next dummy index to hide the fact it was found
  - If not found, get data from permuted memory and put it in the shelter

Why this works: from an observer's perspective, you always do the same thing each time: Linear access **N½** cell in the shelter, and then perform 1 random access to the permuted memory

Oblivious RAM: Square Root Algorithm



After $N^{1/2}$ requests, this current epoch ends → prepare for the next epoch:

- Reshuffle the permuted memory
- How? obliviously updating it with the values in the shelter

Goldreich, Oded, and Rafail 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

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