Advanced Topics in Cryptography

Lecture 3: Private Information Retrieval (PIR), Keyword search

Benny Pinkas

Related papers

- PIR
  - B. Chor, E. Kushilevitz, O. Goldreich, M. Sudan: Private Information Retrieval. J. ACM 45(6): 965-981 (1998)
  - E. Kushilevitz, R. Ostrovsky: Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. FOCS 1997: 364-373

Private Information Retrieval (PIR)

- A special case of secure two-party computation
  - One party (aka sender, server) has a large database.
  - The other party (aka receiver, client) wants to learn a specific item in the database, while hiding its query from the database owner.
  - For example, a patent database, or web access.
- The model:
  - Sender has N bits, $b_1,\ldots,b_N$.
  - Receiver has a query $i \in [1,N]$.
  - Receiver learns b_i (and possibly additional information)
  - Sender learns nothing.
  - The communication is sublinear, i.e. o(N).
- (This model is not very realistic, but is convenient since it's the most basic form of PIR)

Simple protocols

1. Receiver sends i to sender
   – No privacy.

• Sender sends the whole database to the receiver
   – Best privacy for the receiver.
   – Communication is O(N).

• Receiver hides its real question among other randomly chosen questions
   – Sends $i_1,\ldots,i_m$, where there is a j s.t. $i_j=i$, and $m<N$.
   – Sender returns the corresponding m bits of its database.
   – There is some privacy, but the sender can find i with probability 1/m (possibly even with better probability).

How is PIR different from OT (oblivious transfer)?

- PIR
  - Sender learns nothing about the query (i.e., about i).
  - Receiver might learn more than the item it is interested in (b_i).
  - Communication is sublinear in N.
  - Requires either O(N) public key operations, or multiple senders.

- 1-out-of-N Oblivious transfer
  - Sender learns nothing about the query (i.e., about i).
  - Receiver learns nothing but the result of its query (b_i).
  - Communication can be linear in N.
  - Best implementation requires log(N) public key operations.

Results

- Unconditional security
- Unconditional privacy, with a single server, requires $\Omega(N)$ communication [CGKS]
  - A communication *c=(x,i)* is *possible* if for a database x and user interested in i there is a positive probability for c.
  - Fix i, and assume that, considering all possible values of the database, the number of possible *c* is smaller than 2^N.
  - Therefore there are (x,i) and (y,i) s.t. *c* is possible for both.
  - By the privacy requirement, *c* must be possible for every *(x,j)*, and similarly for every *(y,j)*.
  - There is a *j* for which $x_j \neq y_j$.
  - But *c* is possible for both *(x,j)* and *(y,j)*. A contradiction!

Results

- Unconditional security
  - consider a setting where
    - $k \geq 2$ servers know the database
    - Servers do not collude. No single server learns about i.
    - The client can send different queries to different servers
- Results [CGKS and subsequent work]
  - 2 servers: $O(N^{1/3})$ communication
  - K servers: $O(N^{1/\Omega\{k\}})$ communication
  - log N servers: Poly(log(N)) communication.

Two-server PIR

- Best result: $N^{1/3}$ communication. We will show a protocol with $N^{1/2}$ communication.

- There is a simple protocol with O(N) communication:
  - Receiver picks a random vector V_0 of length N.
  - It sets V_1 to be equal to V_0, except for the bit in location i, whose value is reversed.
  - It sends V_0 to P_0, and V_1 to P_1.
  - $Server_0$ sends to R a bit c^0, which is the xor of the bits b_i, for which the corresponding bit in V_0 is 1, namely $\sum V_{0,i} b_i$.
  - $Server_1$ sends a bit c^1, computed using V_1.
  - The receiver computes $b_i = c^0 \oplus c1$.
  - Privacy: Each server sees a random vector.

Two-server PIR with $O(N^{1/2})$ communication

- Suppose $N = m \times m$.
- Database is $\{ b_{i,j} \}_{1 \leq i,j \leq m}$
- Receiver is interested in $b_{\alpha,\beta}$
- picks a random vector V_0 of length m.
- V_1 is V_0 with bit α reversed
- Sends V_0 to S_0 and V_1 to S_1
- S_0 computes and sends the corresponding xor of every column: $c^0_j = \oplus_{i=1\ldots m} V_{0,i}\, b_{i,j}$ (m results in total)
- S_1 computes and sends similar values c^1_j with V_1
- The receiver ignores all values but c^0_β, c^1_β. Computes $b_{\alpha,beta} = c^0_\beta \oplus c^1_\beta$ (but can also compute all $b_{\alpha,j}$)

Four-server PIR with $O(N^{1/2})$ communication

- Here receiver can only compute $b_{\alpha,\beta}$ (and some additional xors of inputs)
- Four servers, $S_{0,0}, S_{0,1}, S_{1,0}, S_{1,1}$. Each sends only O(1) bits

- Database is $\{ b_{i,j} \}_{1\le i,j \le m}$. Receiver is interested in $b_{\alpha,\beta}$.
- Receiver picks random V^R_0, V^C_o of m bits each. Computes V^R_1, V^C_1 by reversing bit α in V^R_0, and bit β in V^C_o.
- Sends vectors V^R_0, V^C_0 to $S_{0,0}$, vectors V^R_0, V^C_1 to $S_{0,1}$, etc.

- Each $S_{a,b}$ computes the xor of the bits whose coordinates correspond to "1" values in V^R_a, V^C_b, and returns the result.
- The receiver computes the xor of the bits it receives…

Four-server PIR with $O(N^{1/3})$ communication

- We showed a four-server PIR where the receiver sends $O(N^{1/2})$ bits and each server send $O(1)$ bits.
- We can use this protocol as a subroutine:
  - Given a database of size N, divide it to $N^{1/3}$ smaller databases of size $N^{2/3}$ each.
  - Apply the previous protocol to all of them in parallel. The receiver constructs sets V^R, V_C for the database which includes the bit it is interested in, and uses these sets for all databases.
  - The receiver sends $O((N^{2/3})^{1/2})=O(N^{1/3})$ bits.
  - Each sender returns $N^{1/3} \cdot O(1) = O(N^{1/3})$ bits.
  - The receiver learns one value from every database.

Computational PIR [KO]

- Security is not unconditional, but rather depends on a computational assumption about the hardness of some problem
- Enables to run PIR with a single server (unlike the infeasibility result for unconditional PIR)

Computational PIR

- We will show computational PIR based on the existence of Homomorphic encryption
- Homomorphic encryption
  - Public key encryption
  1. Given E(x) it is possible to compute, without knowledge of the secret key, E(c·x), for every c.
  2. Given E(x) and E(y), it is possible to compute E(x+y)

- We actually need a weaker property
- Can be implemented based on the hardness of Quadratic Residousity, ElGamal encryption, etc.

Computational PIR: basic scheme

- Suppose $N = s \times t$.
- Database is $\{ b_{i,j} \}_{1\leq i \leq s,\ 1\leq j \leq t}$
- Receiver is interested in $b_{\alpha,\beta}$
- Receiver computes a vector V of size t: $(E(e_1),\ldots,E(e_t))$, where $e_j=0$ if $j\neq \beta$, and $e_\beta=1$.
- Receiver sends V to sender.
- Sender computes, for every row $1\leq i \leq s$,
  $c_i = \sum_{j=1}^{t} E(e_j\cdot b_{i,j}) = E(\sum_{j=1}^{t} e_j\cdot b_{i,j}) = b_{i,\beta}$ $(O(N)$ exponen.)
- Sender sends $c_1,\ldots,c_s$ to receiver. Receiver learns c_α.
- Setting $s=t=N^{1/2}$ results in $O(N^{1/2})$ communication.
- Can we do better?

Computational PIR: reducing the communication via recursion

- In the final step the sender sends s values, while the receiver is interested in only one of them.
  - They can run a PIR in which the receiver learns this value!
- Set $t=N^{1/3}$. Run the previous protocol without the final step.
  - $O(t)=O(N^{1/3})$ communication for this step.
  - At the end of the protocol the sender has $N_1=N^{2/3}$ values (each of length k, which is the length of the encryption).
  - The parties run the previous protocol k times (for each bit of the answers) with $s=t=(N_1)^{1/2}=N^{1/3}$.
  - Communication: $R \Rightarrow S$: $kN^{1/3}+k^2N^{1/3} = O(N^{1/3})$
  - $S \Rightarrow R$: $k^2N^{1/3} = O(N^{1/3})$

Computational PIR: continuing the recursion

- Start from $t = N^{1/4}$.
- There are $N^{3/4}$ answers, each of length k.
- Run the previous protocol on these answers, once for every bit of the answer (a total of k times).
  - The communication overhead is $O(k^3 N^{1/3})$ bits.

- In the general case
  - The recursion has L steps
  - Start from $t=N^{1/(L+1)}$
  - The total communication is $O(N^{1/(L+1)} \cdot k^L)$
  - Setting $L=O((\log N / \log k)^{1/2})$ results in $N^{1/(L+1)} = k^L$, and total communication $2^{O(\sqrt{(\log N / \log k)})}$

- There is another PIR protocol with polylogN comm.

Sender privacy

- PIR does not prevent receiver from learning more than a single element of the database.

- PIR
  - Sender learns nothing about the query (i.e., about i).
  - Receiver might learn more than the item it is interested in (b_i).
  - Communication is sublinear in N.

- 1-out-of-N Oblivious transfer
  - Sender learns nothing about the query (i.e., about i).
  - Receiver learns nothing but the result of its query (b_i).
  - Communication can be linear in N.

- Is it possible to get the best in both worlds?

Symmetrical PIR (SPIR)

- SPIR is PIR with sender privacy:
  - Sender learns nothing about the query (i.e., about i).
  - Receiver learns nothing but the result of its query.
  - Communication is sublinear in N.

- OT + PIR = SPIR
  - Recall 1-out-of-N OT:
    - 2logN keys are used to encrypt N items.
    - Receiver uses logN invocations of OT to learn logN keys.
    - All N encrypted items are sent to the receiver, who can decrypt on of them.
    - The last step can be replaced by PIR.

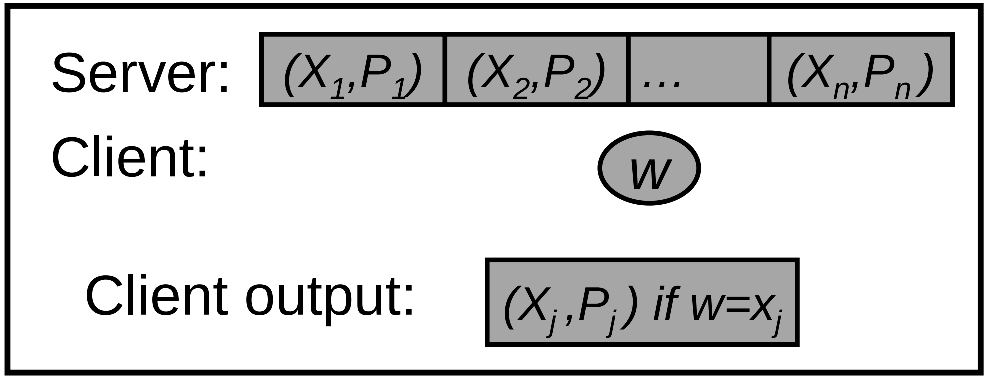
Keyword search

- Motivation: sometimes OT or PIR arenot enough
- Bob:
  - Has a list of *N* numbers of fraudulent credit cards
  - His business is advising merchants on credit card fraud
- Alice (merchant):
  - Received a credit card c, wants to check if it's in Bob's list
  - Wants to hide card details from Bob

- Can they use oblivious transfer or PIR?
  - Bob sets a table of $N=10^{16} \approx 2^{53}$ entries, with 1 for each of the m corrupt credit cards, and 0 in all other entries.
  - Run an oblivious transfer with the new table…
  - …but Bob's list is much shorter than 2^{53}

Keyword Search (KS): definition

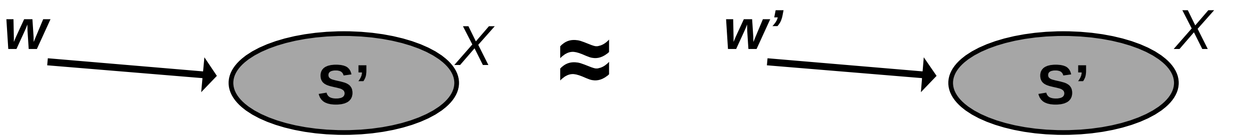
- Input:
  - Server/Bob $X=\{ (x_i,p_i) \}, 1 \le i \le N.$
    - x_i is a keyword (e.g. number of a corrupt credit card)
    - p_i is the payload (e.g. explanation why the card is corrupt)
  - Client/Alice: w (search word) (e.g. credit card number)



- Output:
  - Server: nothing
  - Client:
    - p_i *if* $\exists i$ s.t. $x_i=w$
    - nothing otherwise
- Privacy: Server learns nothing about w, Client learns nothing about (x_i,p_i) for $x_i \neq w$

Keyword Search: Privacy

- Client privacy:
  - (*indistinguishability*) ∀server program S', ∀ *X,w,w', the views of S' in the protocol on server input* X, *for client inputs w* and *w'*, are computationally indistinguishable.



- Server privacy:
  - (*comparison with ideal model*) ∀ client program C', there is a client program C'' in the *ideal model*, s.t. ∀ *(X,w)* the outputs of C' and C'' are computationally indistinguishable.

Specific KS protocols using polynomials

- Tool: Oblivious Polynomial Evaluation (OPE) [NP]
  - Server input: $P(x) = \Sigma_{i=0 \ldots d}\, a_i x^i$, polynomial of degree d.
  - Client Input: w.
  - Client's output: $P(w)$
  - Privacy: server doesn't learn anything about w. Client learns nothing but $P(w)$.
  - Common usage: source of $(d+1)$-wise independence.

- Implementation based on homomorphic encryption
  - Homomorphic encryption: Given $E(x), E(y)$, can compute $E(x+y), E(c \cdot x)$, even without knowing the decryption key.
  - Client sends $E(w), E(w^2), \ldots, E(w^d)$.
  - Sender returns $\Sigma_{i=0 \ldots d}\, E(a_i w^i) = E(\Sigma_{i=0 \ldots d}\, a_i w^i) = E(P(w))$.

KS using OPE (basic method)

- Server's input $X=\{(x_i,p_i)\}$.
- Server defines
  - Polynomial $P(x)$ s.t. $P(x_i)=0$ for $x_i \in X$. (degree = N)
  - Polynomial $Q(x)$ s.t. $Q(x_i)= p_i | 0^k$ for $x_i \in X$. *(k=20?)*
  - $Z(x) = r \cdot P(x)+Q(x)$, with a random r.
    - $Z(x) = p_i | 0^k$ for $w \in X$
    - $Z(w)$ is random for $w \notin X$

- Client/server run OPE of $Z(w)$
  - If $w \notin X$ client learns nothing
  - If $w \in X$ client learns p_i
  - Overhead is $O(N)$

Reducing the Overhead using Hashing

- Server
  - defines $L=N^{1/2}$ bins, maps L inputs to every bin (arbitrarily). (Essentially defines L different databases.)
  - Defines polynomial Z_j for bin j. (Each Z_j uses a different random coefficient r for $Z_j(x) = r \cdot P_j(x)+Q_j(x)$.)
- Parties do an OPE of L polynomials of degree L.
  - Compute $Z_1(w), Z_2(w), \ldots, Z_L(w)$,
- Overhead:
  - $O(L)=O(N^{1/2})$ communication.
  - $O(N)$ computation at the server.
  - $O(L)=O(N^{1/2})$ computation at the client.

Reducing the overhead using PIR (slightly more theoretical…)

- Server:
  - Defines $L= N / \log N$ bins, and uses a *public* hash function H, *chosen independently of* X, to map inputs to bins.
  - Whp, at most $m=O(\log(N))$ items in every bin.
  - Therefore, define polynomials of degree m for every bin.
- Client:
  - Does, in parallel, an OPE for all polynomials.
  - Server has intermediate results $E(Z_1(w)),\ldots,E(Z_L(w))$.
  - Uses PIR to obtain answer from bin $H(w)$, i.e. $E(Z_{H(w)}(w))$.
- Overhead:
  - Communication: logN + overhead of PIR. A total of polylog(N) bits.
  - Client computation is $O(m)=O(\log N)$
  - Server computation is $O(N)$