Fast Private Set Intersection from Homomorphic Encryption

許博翔

May 15, 2024

Outline

Outline

Private Set Intersection

- Public: N_X, N_Y, σ
- Sender: $X \subseteq \{0,1\}^\sigma$ with size N_X
- Receiver: $Y \subseteq \{0,1\}^\sigma$ with size N_Y
- Goal:
  - Sender should not know anything about Y.
  - Receiver should know $X \cap Y$.
  - Receiver should not know anything about $X \setminus Y$.
- Threat model: semi-honest security model (both parties correctly follow the protocol, but may try to learn as much as possible from their view of the protocol execution)

Fast Private Set Intersection

- Original communication complexity: $O(N_X N_Y)$
- Goal communication complexity: $O(N_Y \log N_X)$
- The protocol works for all N_X, N_Y, but since it's powerful when $N_X \gg N_Y$, assume that $N_X \gg N_Y$.

FHE (Fully Homomorphic Encryption)

- Homomorphism: $\varphi : A \to B$ with $\varphi(x \circ_A y) = \varphi(x) \circ_B \varphi(y)$.
- Homomorphic Encryption: $\mathrm{Encrypt}, \mathrm{Decrypt}$ are homomorphisms.
- Types of homomorphic encryption: (distinguish by the arithmetic circuits they support)
  - PHE (partially homomorphic encryption): one type of gates with unlimited depth
  - SHE (somewhat homomorphic encryption): two types of gates with limited depth
  - Leveled fully homomorphic encryption: multiple types of gates with limited depth
  - FHE (fully homomorphic encryption): multiple types of gates with unlimited depth

IND-CPA Secure

- IND-CPA: indistinguishability under chosen-plaintext attack
- Steps:
  - Challenger: Generate (pk, sk).
  - Adversary: Choose and send m_0, m_1 to the challenger.
  - Challenger: Uniformly randomly choose $b \in \{0, 1\}$, and send $\mathrm{Encrypt}(m_b, pk)$ back to the adversary.
  - Adversary: Submit a guess for b.
- Restriction: The adversary can only perform polynomially bounded number of operations.
- Goal: The adversary's guess is correct with probability $\frac{1}{2} + \mathrm{negl}(\lambda)$, where λ is the security parameter.

Assumption

- $\mathrm{FHE.Encrypt}, \mathrm{FHE.Decrypt}$: Encryption and decryption of a IND-CPA secure FHE scheme
- Threat model: semi-honest security model
- $N_X \gg N_Y$

Outline

The Basic Protocol

- $t \in \mathbb{P}$ is large enough to encode $\{0,1\}^\sigma$ as elements of $\mathbb{Z}_t$.
- Receiver:
  - Generate (pk, sk).
  - Send $(c_1, c_2, \ldots, c_{N_Y})$ to sender, where $Y = \{y_1, \ldots, y_{N_Y}\}$ and $c_i = \mathrm{FHE.Encrypt}(y_i, pk)$.
- Sender:
  - Uniformly randomly sample $r_i \in \mathbb{Z}_t^*$.
  - Homomorphically compute $d_i = r_i \prod_{x \in X}(c_i - x)$.
  - Return $(d_1, d_2, \ldots, d_{N_Y})$ to receiver.
- Receiver: $X \cap Y = \{y_i : \mathrm{FHE.Decrypt}(d_i, sk) = 0\}$.

The Basic Protocol

- $\mathrm{FHE.Decrypt}(d_i, sk) = r_i \prod_{x \in X}(y_i - x)$.
- If $y_i \in X$, then $\mathrm{FHE.Decrypt}(d_i, sk) = 0$.
- If $y_i \notin X$, then $\mathrm{FHE.Decrypt}(d_i, sk)$ is a uniform distribution on $\mathbb{Z}_t^*$, independent of $\prod_{x \in X}(y_i - x)$.
- $O(N_X N_Y)$ homomorphic multiplications and additions

Outline

Batching

- Goal: Operate on n items simultaneously.
- $R := \mathbb{Z}[x]/(x^n+1), R_t := R/tR$, where n is a power of 2
- $R_t \cong \mathbb{Z}_t^n$ for suitable t
- SIMD (single instruction, multiple data): plaintext space $\mathbb{Z}_t^n$
- Receiver: Group Y into $\frac{N_Y}{n}$ vectors of length n, and encrypt the vectors to $c_1, c_2, \dots$.
- Sender: Homomorphically compute $d_i = r_i \prod_{x \in X}(c_i - x)$, where $r_i \in (\mathbb{Z}_t^*)^n$.

Hashing

- Hashing d items into a hash table of size d results in a maximum load of $O(\log d)$ with high probability.
- Proof:
  Let $k = \log d \gg e^3$, and $\epsilon > 0$.
  $k(\log e - \log k) < k(\log e - \log e^3) < -2k < \log \epsilon - \log d$.
  $\Rightarrow \mathbb{P}\{$ maximum load exceed $k\} \leq \binom{d}{k}(\frac{1}{d})^{k-1} < (\frac{de}{k})^k(\frac{1}{d})^{k-1} = d(\frac{e}{k})^k < \epsilon$.
- Hash X, Y into d bins, and run PSI for each bin.
- Uneven loads reveal additional information $\Rightarrow$ Every bin must be padded to a fixed size $\Rightarrow$ Receiver and sender set two different dummy values from $\mathbb{Z}_t$ that are not legitimate values, and use them to pad the bins.
- Complexity: $O(d \log^2 d)$

Cuckoo Hashing

- Cuckoo hashing:
  - $h > 1$ hash functions $H_1, \ldots, H_h$
  - To insert x, randomly choose $i \in [h]$ and insert (x, i) at location $H_i(x)$. If this location was already occupied by (y, j), remove (y, j) and reinsert (y, j') where $j' \in [h]$ is chosen randomly.
- Application to our protocol:
  - Number of bins: m, $m \approx N_Y$, $m > N_Y$
  - Receiver: Perform cuckoo hashing.
  - Sender: Perform normal hashing, and insert all hN_X elements of $[h] \times X$.
  - Assume $hN_X > m \log m$.
  - $\mathbb{P}\{$ at least one bin has load $> B\} \leq m \sum_{i=B+1}^{d} \binom{d}{i} (\frac{1}{m})^i (1 - \frac{1}{m})^{d-i}$.
  - B is upper-bounded by $\frac{d}{m} + O(\sqrt{\frac{d \log m}{m}})$ with high probability.

Permutation-based Hashing

- Suppose that m is a power of 2.
- $x \to x_L \| x_R$, where x_R is of length $\log_2 m$.
- Location function $\mathrm{Loc}_i(x) := H_i(x_L) \oplus x_R$.
- Insert (x_L, i) to $\mathrm{Loc}_i(x)$.
- Receiver: Perform the insertion of cuckoo hashing.
- Sender: Perform the insertion of normal hashing.
- Correctness: If $(x_L, i) = (y_L, j)$ and $\mathrm{Loc}_i(x) = \mathrm{Loc}_j(y)$, then $x = y$.
- Reduce the length of the strings stored in the hash table by $\log_2(m) - \lceil \log_2(h) \rceil$.

Hashing to A Smaller Representation

- Usually, $N_X + N_Y \longleftarrow 2^{\sigma} \Rightarrow$ Hash $N_X \cup N_Y$ to $2^{\sigma_{\max}}$.
- Probability of a collision $\leq \binom{N_X+N_Y}{2} \times 2^{-\sigma_{\max}} < (N_X + N_Y)^2 \times 2^{-\sigma_{\max}-1}$.
- Want: Probability of a collision $\leq 2^{-\lambda}$.
- $\Rightarrow \sigma_{\max} \geq 2\log_2(N_X + N_Y) + \lambda - 1$.
- Combine with permutation-based hashing: $\sigma_{\max} - \log_2 m + \lceil \log_2 h \rceil$
- Choose t s.t. $\log_2 t > \sigma_{\max} - \log_2 m + \lceil \log_2 h \rceil + 1$ is enough.
- Combine with batching:
  - Receiver: $\frac{m}{n}$ plaintext vectors
  - Sender: $\frac{Bm}{n}$ plaintext vectors

Reducing the Circuit Depth - Windowing

- Recall: Compute the encryption of $r\prod_{x\in X}(y-x) = ry^{N_X} + ra_{N_X-1}y^{N_X-1} + \cdots + ra_0$.
- Original:
  - Receiver sends the encryption of y.
  - Computing ry^{N_X} needs a circuit of depth $\lceil \log_2(N_X+1) \rceil$.
- Modified:
  - Receiver sends $c^{(i,j)} = \mathrm{FHE.Encrypt}(y^{i2^{\ell j}})$ for all $1 \leq i \leq 2^{\ell} - 1, 0 \leq j \leq \lfloor \frac{\log_2(N_X)}{\ell} \rfloor$.
  - Worst case: A product of $\lfloor \frac{\log_2(N_X)}{\ell} \rfloor + 1$ terms
  - $\Rightarrow$ Needs a circuit of depth $\lceil \log_2(\lfloor \frac{\log_2(N_X)}{\ell} \rfloor + 1) \rceil$.
- ℓ: A computation-communication trade-off

Reducing the Circuit Depth - Partitioning

- Partition X into α subsets.
- Compute $r\prod_{x\in X_1}(y-x), r\prod_{x\in X_2}(y-x), \ldots, r\prod_{x\in X_\alpha}(y-x)$ instead.
- Circuit depth: $\lceil \log_2(\frac{N_X}{\alpha}+1)\rceil$
- Combine with windowing and all of the hashing optimizations above, the circuit depth becomes $\lceil \log_2(\lfloor \frac{\log_2(\frac{B}{\alpha})}{\ell}\rfloor + 1)\rceil + 1$.

Reducing Reply Size via Modulus Switching

- Change the encryption parameter from q to q' if q' is not too small.
- Ciphertext sizes are reduced by a factor of $\frac{\log q}{\log q'}$.

Thank You for Listening