Privacy Preserving Data-Mining

and Its Application to Large-Scale Network Measurements

Matthew Roughan
<matthew.roughan@adelaide.edu.au>

School of Mathematical Sciences
University of Adelaide

Joint work with Yin Zhang <yzhang@cs.utexas.edu>

Data is key

- Data is the key understanding traffic
- Data is the key to good models
- Data is the key to prediction/planning, anomaly detection, traffic engineering, ...
- Good data is hard to get
  - for most people (if not Google)

Why is data hard to get?

- Companies don't share
  - companies don't want to reveal data
    - afraid of misuse of data
    - afraid it will reveal business secrets
    - afraid it will reveal incompetence
  - sometimes they are not allowed to
    - e.g. privacy legislation [?]
- no particular company sees all the Internet
  - the Internet is (by its nature) distributed
  - each company can add a perspective

What's the problem

- How much traffic is there on the Internet?
  - the argument is made [?] that lack of such data contributed to the tech-wreck
  - regulators need such information
    - e.g. anti-trust cases
- Detecting distributed attacks
  - DDoS (Distributed Denial of Service), Worms/viruses,
  - e.g. Worms are easy to detect once they are well under way, but if you want to detect it early, the more data points you have the better.
- but if companies won't share data, how can we collect Internet wide measurements?

Similar problems elsewhere

- The Center for Disease Control and Prevention (CDC) who have to detect new health threats
  - need data from
    - hospitals
    - insurance companies, airlines, ...
    - NGOs (e.g. charities)
    - other government bodies
  - data is
    - proprietary (e.g. insurance risks)
    - protected by privacy legislation
  - data-mining community has developed solutions
    - secure-distributed computing [?, ?, ?]
    - privacy-preserving data-mining [?, ?]

Trusted third party

- simple answer: a trusted third party
  - independent party (e.g. with no vested interest)
  - trusted by all other parties
  - collects data, and shares the results
- problems:
  - hard to find such parties
    - in Internet research anyway
  - an exception is the ABS [?]
    - Australian Bureau of Statistics collects information about ISPs, including some traffic measurements
  - often requires special legislation
    - lacks flexibility

A Couple of problems

Well known problems in the area

- Dining cryptographers
- Millionaire problem

Internet measurement problems

- traffic
- performance

Dining cryptographers

- N cryptographers are having dinner
- When it is time to pay the bill, the waiter tells them that someone has already paid
- the cryptographers are suspicious by nature (particularly Alice and Bob).
  - they suspect the NSA has paid
- not wanting to be compromised by such an association, they need to find out if someone at the table paid, or an external party such as the NSA
- how can they do so, without anyone revealing whether they paid or not?
  - of course, the waiter is sworn to secrecy

Millionaire problem

- Bill Gates and Warren Buffet are trying to decide who should put more money into the Gates foundation $(*)$
  - they want to know who is richer
- But they are feeling rather secretive, and don't want to reveal their true wealth.
- how can they decide?

Primitives

There are some generic techniques that can help us out

- Secure Distributed Summation (SDS)
- Secure distributed dot product
- Oblivious transfer

Secure Distributed Summation

Problem: N parties each have one value v_i and they want to compute the sum

$$V = \sum_{i=1}^{N} v_i$$

but they don't want any other party to learn their value.

SDS algorithm [?]

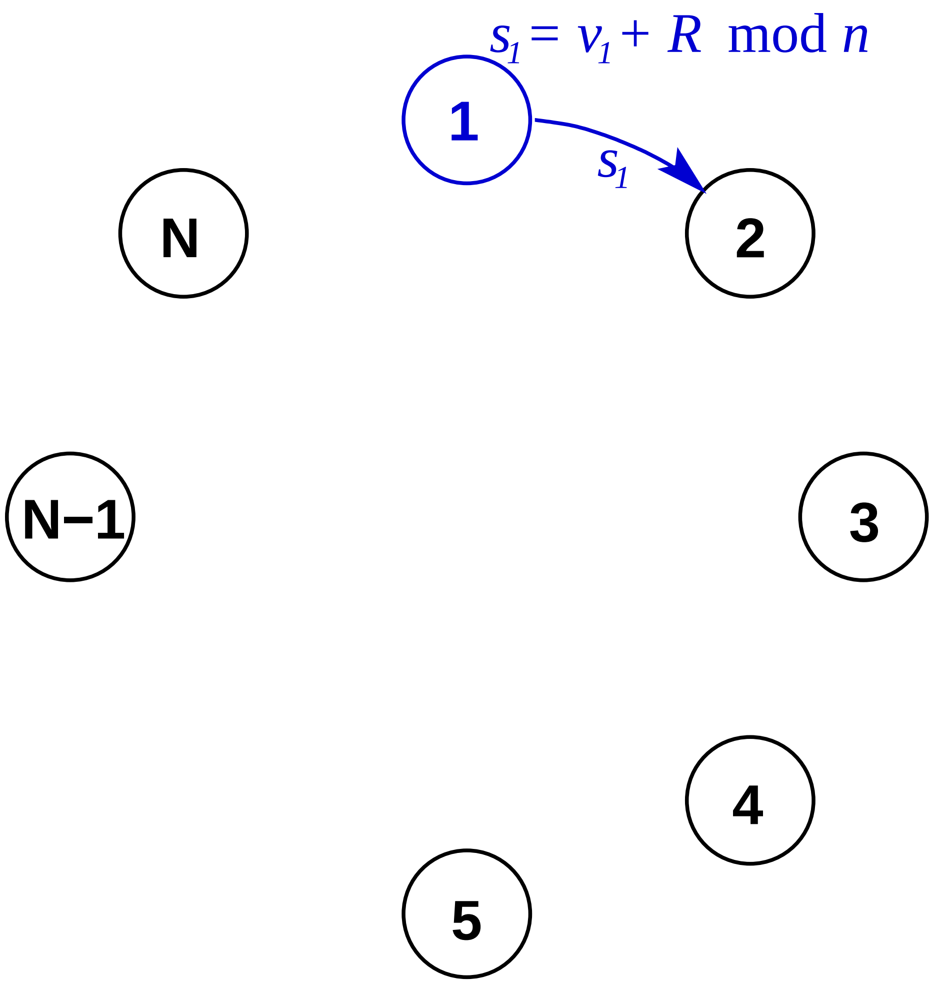
Assume the value $V \in [0,n]$ (for large n)

```
party 1:  randomly generate R ~ U(0,n)
party 1:  compute s_1 = v_1 + R mod n
party 1:  pass s_1 to party 2
for i=2 to N
  party i:  compute s_i = s_{i-1} + v_i mod n
  party i:  pass s_i to party i+1
endfor
party 1:  compute v_N = s_N - R mod n
```

Finally, party 1 has to share the result with the others.

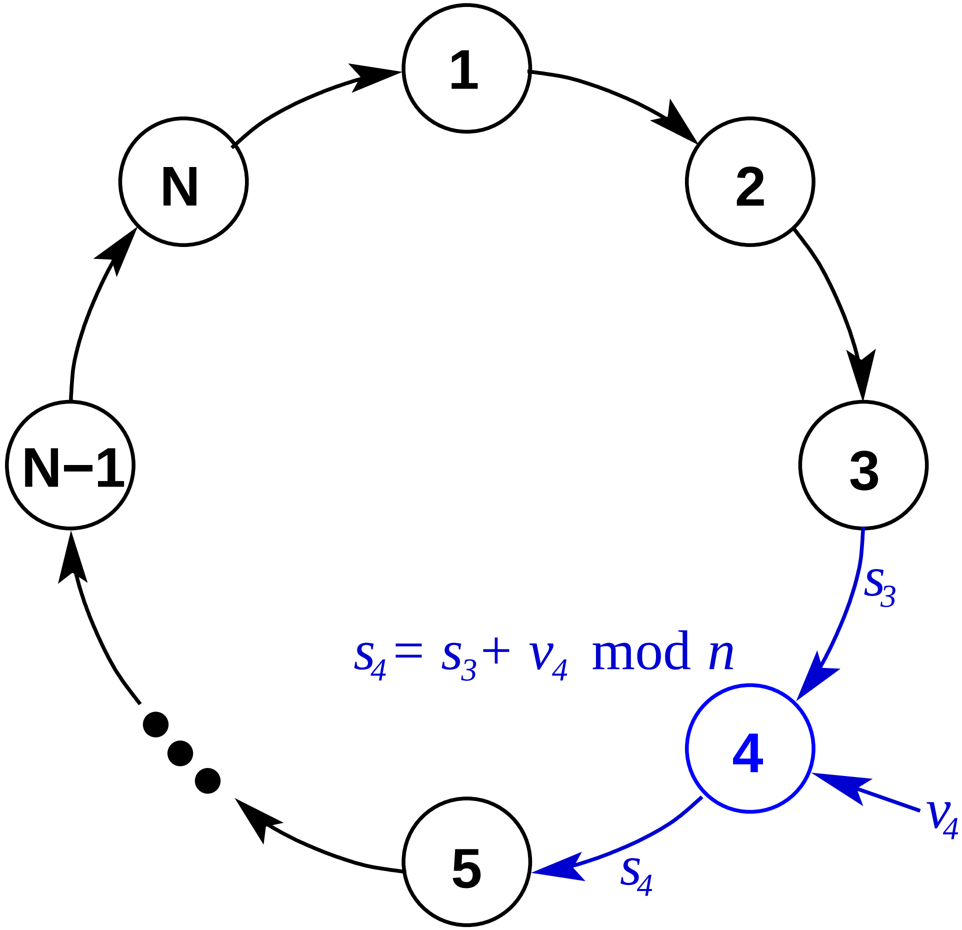
s_i will be uniformly randomly distributed over $[0,n]$ and so we learns nothing about any other parties values.

SDS algorithm



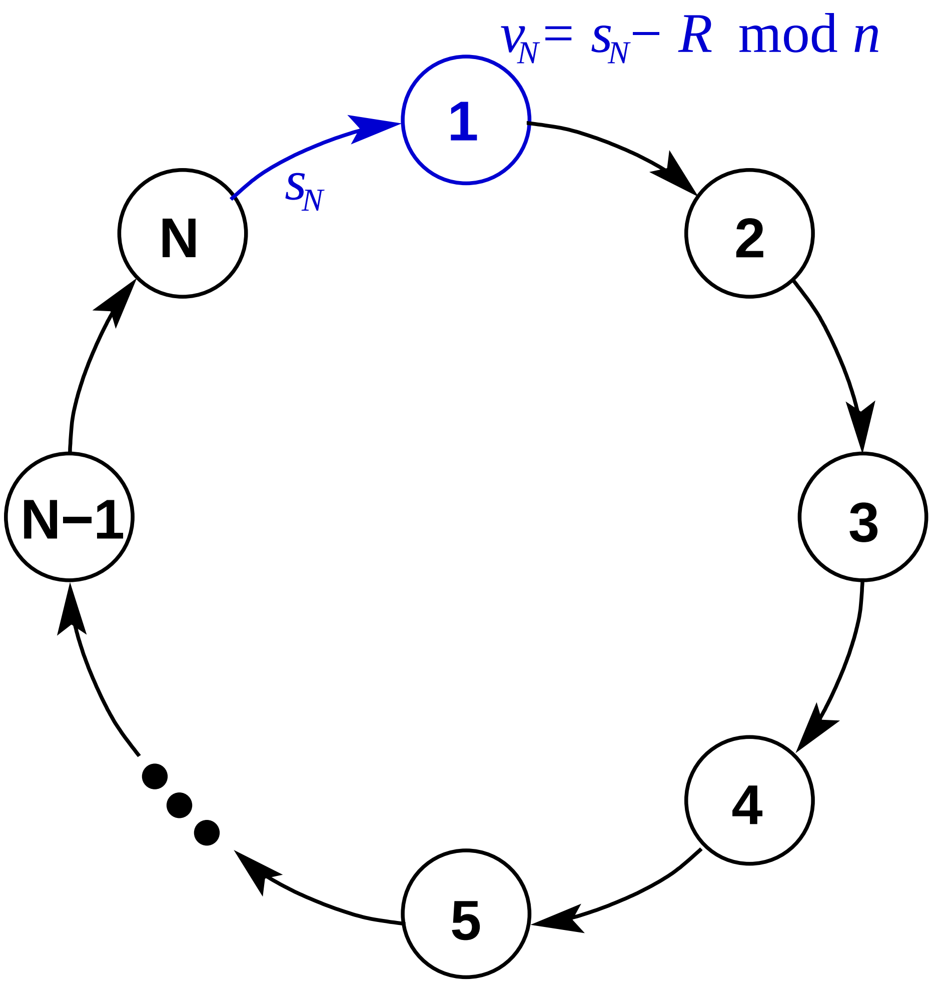

```
party 1:  randomly generate R ~ U(0,n)
party 1:  compute s_1 = v_1 + R mod n
party 1:  pass s_1 to party 2
for i=2 to N
   party i:  compute s_i = s_{i-1} + v_i mod n
   party i:  pass s_i to party i+1
endfor
party 1:  compute v_N = s_N - R mod n
```

SDS algorithm




```
party 1:  randomly generate R ~ U(0,n)
party 1:  compute s_1 = v_1 + R mod n
party 1:  pass s_1 to party 2
for i=2 to N
   party i:  compute s_i = s_{i-1} + v_i mod n
   party i:  pass s_i to party i+1
endfor
party 1:  compute v_N = s_N - R mod n
```

SDS algorithm




```
party 1:  randomly generate $R \sim U(0,n)$
party 1:  compute $s_1 = v_1 + R \mod n$
party 1:  pass $s_1$ to party 2
for i=2 to N
   party i:  compute $s_i = s_{i-1} + v_i \mod n$
   party i:  pass $s_i$ to party $i+1$
endfor
party 1:  compute $v_N = s_N - R \mod n$
```

Applications

- dining cryptographers
  - v_i equals 1 if a diner paid, zero otherwise, $n = 1$, and $V \in \{0, 1\}$
- calculating the total traffic on the Internet
  - v_i is total per ISP
  - need some care to avoid double-counting
- Internet health (e.g. by accumulating certain statistics, e.g. packet drops)
  - e.g. v_i is packet loss percent at each ISP
  - use sum to compute (weighted) average
  - time series algorithms (either pre- or post-)
- Sketches

Application to Sketches

Could apply this approach to many sources of data

- number of routers, number of links, or number of links of each type (e.g. OC48, Gig-Ethernet)
- kilometres of fiber, bandwidth-miles of network capacity,
- traffic-miles for carried traffic,
- detailed traffic data (e.g. netflow)
- performance data (packet loss, delay, reordering, ...)

Lots of sorts of data, and in particular for complex data (traffic) the dimensionality of dataset could be very high.

Application to Sketches

Sketches [?] are an approach to reduce dimensionality of streaming datasets, e.g. Count-Min sketch [?]

- **Data:** a stream of updates (a,u), where $a \in \{1,\ldots,n\}$ is a key, and $u \in \mathbb{R}$ a value.
- **Signal:** a vector $v \in \mathbb{R}^n$, where for each update (a,u), we perform $v_a {+}{=} u$.
- **Sketch:** consists of a $d \times w$ array of counts: $c[1,1] \ldots c[d,w]$, and d random hash functions $h_1, \cdots, h_d : \{1 \cdots n\} \rightarrow \{1 \cdots w\}$, for $w \ll n$
- **Update:** When an update (a,u) arrives, update $c[i,h_i(a)] {+}{=} u$ for all $1 \leq i \leq d$.
- **Query:** When a point query $Q(a)$ arrives, an approximation of v_a is given by $\hat{v}_a = \min_i c[i,h_i(a)]$.

Application to Sketches

Its almost trivial to extend SDS to sketches:

- agree on common hash functions (and array sizes)
- compute a sketch locally at each party
- use SDS to sum each element in the array
  - the point is that given K updates $\{(a_i^{(n)}, u_i^{(n)})\}_{i=1}^K$ from party n

$$\text{Sketch}\left(\cup_{n=1}^N \{(a_i^{(n)}, u_i^{(n)})\}_{i=1}^K\right) = \sum_{n=1}^N \text{Sketch}\left(\{(a_i^{(n)}, u_i^{(n)})\}_{i=1}^K\right)$$

- we can use the final sketch as needed, e.g. in anomaly detection

Honest but curious model

- any party could corrupt the total V by inputing incorrect data v_i
- calculation has implicit assumption of honesty
  - let us extend this
- "Honest but curious" security model
  - **honest:** honestly follow protocol
  - **curious:** may perform more operations to try and learn more information (than they were supposed to learn)
- doesn't prevent colluding coalitions
- conditions can be weakened (e.g. honest majority)

Collusion

- Assume party j and $j+2$ collude
  - They know at least s_j and s_{j+1}
  - $s_{j+1} - s_j \mod n = v_j$
  - so they can learn the value of j
- Various methods of prevention, e.g.
  - divide v_i randomly into shares v_{im} such that

$$\sum_m v_{im} = v_i$$

  - sum over i in a different order for each m.

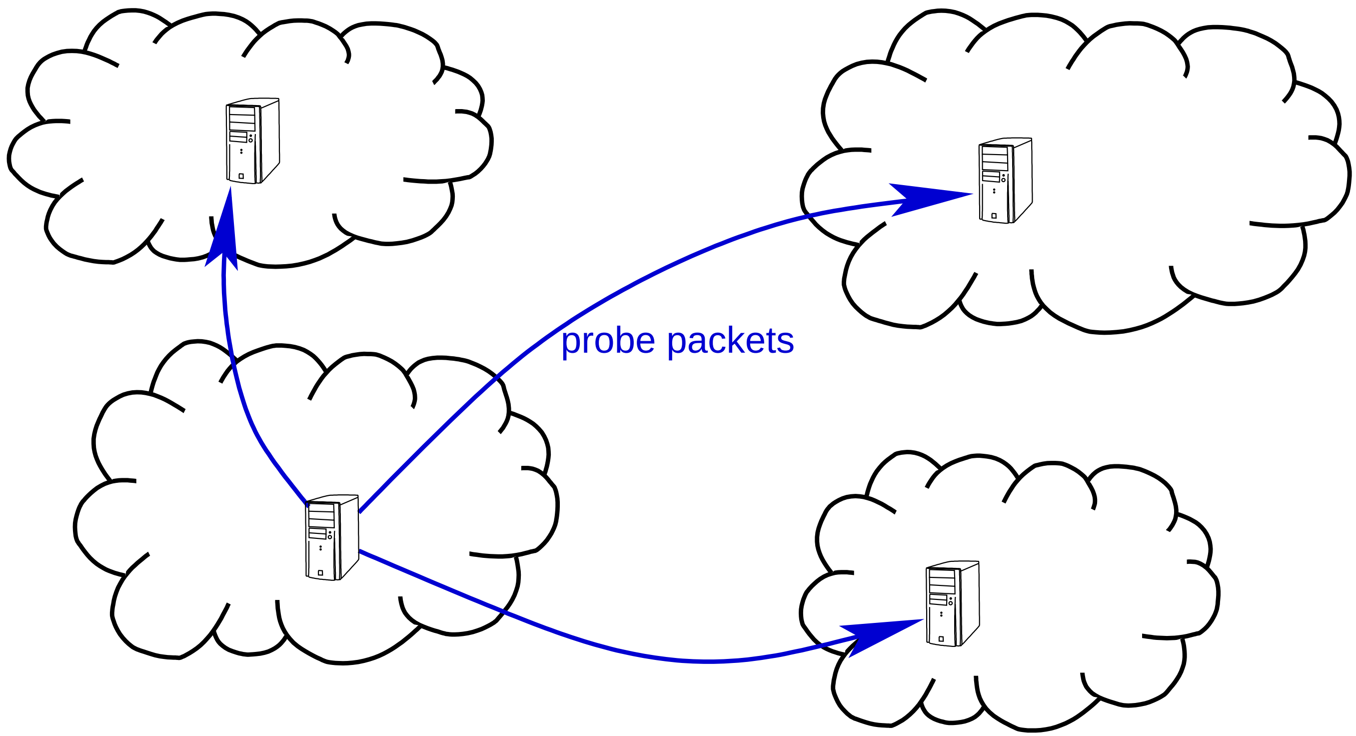
$$\sum_{i=1}^{N} v_{im} = V_m$$

  - sum V_m normally $V = \sum_m V_m$

Another applications

ISPs measure one-way inter-provider performance

- **inter-provider:** many problems occur at the edges
- **one-way:** inter-ISP routing is asymmetric



Internet perf. measurement

Experiment and notation:

- send K_{ij} probe packets from ISP $i \to j$
- sender i notes transmit times $t_{ij}^{(k)}$
- receiver j notes receive times $r_{ij}^{(k)}$
- delay $d_{ij}^{(k)} = r_{ij}^{(k)} - t_{ij}^{(k)}$
- averages:

$$\bar{D}_{ij} = \frac{1}{K_{ij}} \sum_{k=1}^{K_{ij}} r_{ij}^{(k)} - t_{ij}^{(k)}$$

$$\bar{R}_{ij} = \frac{1}{K_{ij}} \sum_{k=1}^{K_{ij}} r_{ij}^{(k)}, \qquad \bar{T}_{ij} = \frac{1}{K_{ij}} \sum_{k=1}^{K_{ij}} t_{ij}^{(k)}$$

Internet perf. measurement

- but what if the ISP's don't want other ISPs to be able to make comparisons?
  - obviously this limits the type of measures we can make: consider averages across providers, e.g.

$$\bar{D}_i^{\text{out}} = \frac{1}{N-1} \sum_{\substack{j=1 \\ j \neq i}}^{N} \bar{D}_{ij}$$

  - limits what data can be shared:
    - ISP's can't share individual measurements $r_{ij}^{(k)}$ or $t_{ij}^{(k)}$

SDS to the rescue

$$
\begin{aligned}
\bar{D}_i^{\text{out}} &= \frac{1}{N-1} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{1}{K_{ij}} \sum_{k=1}^{K_{ij}} \left[r_{ij}^{(k)} - t_{ij}^{(k)} \right] \\
&= \frac{1}{N-1} \left[\sum_{\substack{j=1 \\ j \neq i}}^{N} \bar{R}_{ij} - \sum_{\substack{j=1 \\ j \neq i}}^{N} \bar{T}_{ij} \right]
\end{aligned}
$$

- $\sum_{\substack{j=1 \\ j \neq i}}^{N} \bar{T}_{ij}$ is already known by i
- $\sum_{\substack{j=1 \\ j \neq i}}^{N} \bar{R}_{ij}$ calculate using SDS and give i the result

But wait...

What happens when packet are lost?

- we can't compute $\bar{D}_i^{\text{out}}$ without censoring the transmit times for the lost packets
- we can't tell other ISPs when packet are lost
  - this would reveal a great deal about performance
- we can't include straight sequence numbers in packets
  - these would allow statistical inference

Secure Dot Product (SDP) [?]

- Alice has a vector $\mathbf{a}$, and Bob has a vector $\mathbf{b}$.
- They want to compute

$$\mathbf{a}.\mathbf{b} = \sum a_i b_i$$

without revealing any a_i or b_i to each other

- can't just return $\mathbf{a} \cdot \mathbf{b}$ because some choices of $\mathbf{a}$ would reveal parts of $\mathbf{b}$.
- so split the solution

$$V_a + V_b = \mathbf{a} \cdot \mathbf{b}$$

and return V_a to Alice and V_b to Bob.

Solution

- add a randomly chosen packet ID to each packet:
  - ID chosen randomly from $\{1, 2, \ldots, L\}$ where $L \geq K_{ij}, \forall i, j$
  - create Identity vectors (at receivers)

$$I_{ij}^{(k)} = \begin{cases} 1, & \text{if the packet with ID } k \text{ from } i \text{ to } j \text{ is received,} \\ 0, & \text{otherwise.} \end{cases}$$

- now the calculation is

$$\bar{D}_i^{\text{out}} = \frac{1}{M_i} \sum_{\substack{j=1 \\ j \neq i}}^{N} \left[\sum_{k=1}^{L} I_{ij}^{(k)} r_{ij}^{(k)} - \sum_{k=1}^{L} I_{ij}^{(k)} t_{ij}^{(k)} \right].$$

Solution

- $I_{ij}^{(k)} r_{ij}^{(k)}$ is known to each receiver j, and so the sum (over k) is easily performed, and we can compute the sum over j using a SDS as before
- the sum $\sum_{k=1}^{L} I_{ij}^{(k)} t_{ij}^{(k)}$ is a dot product, and so we use SDS to get two parts of this $s_{ij}^{(t)}$ and $s_{ij}^{(r)}$.
  - $s_{ij}^{(t)}$ goes to the transmitter, and so we can perform a standard sum over j on these
  - $s_{ij}^{(r)}$ goes to the receivers, so we sum using a SDS
- M_i, the total number of received packets (transmitted from i) can be computed using a SDS
- transmitter gets all the info. to compute $\bar{D}_i^{\text{out}}$

Oblivious transfer [?, ?]

- there are various versions
- consider 1-in-n Oblivious Transfer (OT)
  - Alice has a list of numbers $\{a_1, a_2, \ldots, a_n\}$
  - Bob has an index β
  - Bob wants to learn a_β
  - Alice must not learn β, and Bob must not learn a_i for any $i \neq \beta$.
- Bob learns exactly one item from Alice's list, without Alice learning which item Bob discovered.

Applications

- the millionaires problem
  - more generically: calculating a minimum
- Assume Alice has wealth $w_A \in [1, n]$, and Bob has $w_B \in [1, n]$, where n is known to both

Alice creates a list of n numbers

0
0
⋮
0 ← w_B Bob uses 1−in−n OT to obtain the w_B entry
w_A → 1
1
⋮
1

If Bob gets 0
then Bob is poorer
If Bob gets 1
then Bob is at least as rich

Future

- implications for many game theory problems: prisoner's dilemma
  - assumption that lies behind the dilemma: the prisoner's can't trust each other
  - Secure distributed computing provides mechanisms for creating trust
  - end up with more co-operations
- example: inter-ISP traffic engineering

Conclusion

- we can do stuff that I never imagined (until very recently)
- some of it is really cool

$(*)$ – no real millionaires were harmed in the production of these slides

Bonus slides

OT – how it works

1-in-2 Oblivious Transfer

- Alice has a pair of bits (a_0, a_1), and Bob has β
- trapdoor permutation f
  - Given key k, can choose permutation pair (f_k, f_k^{-1})
  - Given f_k it is hard to find f_k^{-1}
  - Easy to choose random element from f_k's domain
- random Bit B_{f_k} is a poly.-time Boolean function
  - $B_{f_k} = 1$ for half of the objects in f_k's domain
    $B_{f_k} = 0$ for other half
  - no probabilistic polynomial time algorithm can make a guess for $B_{f_k}(x)$ that is correct with probability better than $1/2 + 1/\text{poly}(k)$

1-in-2 Oblivious Transfer

- A randomly chooses (f_k, f_k^{-1}), and tells f_k to B
- B randomly chooses x_0 and x_1 in f_k's domain, and computes $f_k(x_i)$
- B sends A the pair

$$(u,v) = \begin{cases} (f_k(x_0), x_1), & \text{if } \beta = 0 \\ (x_0, f_k(x_1)), & \text{if } \beta = 1 \end{cases}$$

- A computes $(c_0, c_1) = \left(B_{f_k}(f_k^{-1}(u), f_k^{-1}(v))\right)$
- A sets $d_i = a_i \text{ xor } c_i$ and sends (d_0, d_1) to B
- B computes $a_\beta = d_\beta \text{ xor } B_{f_k}(x_\beta)$

http://www.cs.ut.ee/~lipmaa/crypto/link/protocols/oblivious.php

SDP - how it works

(1) A and B agree on two numbers m and n

(2) A finds m random vectors $\mathbf{t}_i$ such that

$$\mathbf{a}_1 + \mathbf{a}_2 + \ldots + \mathbf{a}_m = \mathbf{a}$$

B finds m random numbers $r_1, r_2, \ldots, r_m$.

(3) for $i = 1$ to m

(3a) A sends B n different vectors:

$$\{\mathbf{a}_i^{(1)}, \mathbf{a}_i^{(2)}, \ldots, \mathbf{a}_i^{(n)}\}$$

where exactly one $\mathbf{a}_i^{(q)} = \mathbf{a}_i$, the other $n-1$ vectors are random

(3b) B computes $\mathbf{a}_i^{(j)} \cdot \mathbf{b} - r_i$

(3c) A uses 1-in-n OT to retrieve

$$v_i = \mathbf{a}_i^{(q)} \cdot \mathbf{b} - r_i = \mathbf{a}_i \cdot \mathbf{b} - r_i.$$

(4) B computes $V_b = \sum_{i=1}^{m} r_i$

(5) A computes

$$V_a = \sum_{i=1}^{m} v_i = \sum_{i=1}^{m} \mathbf{a}_i \cdot \mathbf{b} - r_i = \mathbf{a} \cdot \mathbf{b} - V_b.$$