Dining Cryptographers Problem

Pranav Nuti and Ishan Agarwal

29th April 2016

The Problem

n cryptographers go out for dinner at a fancy restaurant. When the time comes to pay the (huge) bill, the manager mysteriously informs them that it has been taken care of. The cryptographers are suspicious that the NSA is trying to woo them but they also think that there's a *chance* that one of them is in a generous mood. Of course, they would like to know whether the NSA paid the bill, but if it was one of them who paid, they want that person's identity to remain undisclosed to protect his/her privacy (they are cryptographers after all!). How can they do this?

The Solution

Of course, them being smart cryptographers, they immediately come up with a solution. Cryptographer i and cryptographer j then *privately* decide on a random bit k_{ij} (independent of every other pair). Cryptographer i then announces the value of $b_i = \left(\bigoplus_{j \neq i} k_{ij}\right) \oplus m_i$ where m_i is 1 if cryptographer i had paid for the bill and 0 otherwise. Finally, $\bigoplus_i b_i$ $(= \bigoplus_i m_i)$ is computed. If it 1 it must mean that someone in the group has paid. If it 0, it must mean the NSA has paid.

If the NSA hasn't paid, any group of collaborating non-payer cryptographers S (with size $k < n$) finds that the probability of any non-collaborator paying the bill is precisely $\frac{1}{n-k}$. If this were not so, it would mean that the group is able to obtain (non-trivial) information about $\{m_i | i \notin S\}$ from $\{\left(\bigoplus_{j \neq i, j \notin S} k_{ij}\right) \oplus m_i | i \notin S\}$ when all the keys in the sums are random, independent and unknown (which is of course impossible, the situation being analogous to the one that occurs while using a one time pad). The cryptographers give their procedure a name: protocol 1.

The cryptographers then realize that protocol 1 could be used by a network of n people to communicate with each other while remaining anonymous. All participants agree upon a fixed message length l (they can pad their messages with zeroes if they need to) and they repeat protocol 1-like procedure l times: the unique participant i in the network who has a message $m = r_1 r_2 r_3 \dots r_l$ to broadcast consecutively lets m_i (using notation from above) be $r_1, r_2, \dots, r_l$. Let us call this procedure protocol 2. Protocol 2 lets some one participant broadcast a message of length l bits.

Problems with the Solution

Generating Random Bits

Protocol 2 requires generating a total of $l\binom{n}{2}$ random bits. We can reduce this by using pseudo-random bits determined by a seed (a *key* which each pair of participants agree upon) instead of random bits. The protocols described above would not be perfectly secure, but they would still be computationally secure (and our statements about probability would be off by a *negligible* factor).

But even after doing this we still need to generate $\Theta(n^2)$ random bits for the keys. We can reduce this by removing the condition that every pair of participants must agree upon a random key. Say, for example, that the n participants are seated in a circle and each participant only agrees on a key with his neighbours. This brings the number of random bits down to $\Theta(n)$. However, if the two neighbours of a participant collaborate, they can determine (with probability 1) whether their common neighbour is broadcasting the message. Note that this is unlike what happens when all pairs exchange keys.

This is a general problem with reducing the number of keys. Consider each participant in the network a vertex of a graph G and let the edge set E consist of the pairs of participants who exchange keys. Consider a set $S \subset G$ and

let $N(S)$ be the set of neighbours of S. If all the participants in $N(S) \setminus S$ are collaborating, then they can determine whether the message was sent by a participant in S or not by computing $\bigoplus_{i \in S} m_i = \left(\bigoplus_{i \in S} b_i\right) \oplus \left(\bigoplus_{e \in L} k_e\right)$ where L is the set of edges from S to $N(S) \setminus S$. If the graph G is sparse (and the number of keys generated is low), the size of $N(S) \setminus S$ can be quite small.

Multiple senders and scheduling

As described above, protocol 2 only allows one sender. If multiple people try to send a message, all messages are lost. The only way we can fix this is by running protocol 2 for multiple rounds and then reserving particular rounds for particular participants of the network, so that two people don't broadcast at the same time. But how can the participants agree upon who will broadcast when while preserving anonymity?

To achieve this, the participants in the network first agree upon a large number q, the number of times protocol 2 will be run. They also agree that each participant can broadcast a total of p messages each (of length l each, as discussed earlier). Then they run a special *scheduling* round of protocol 2 before beginning their broadcasts. In this special round, participant i broadcasts $b_i = \left(\bigoplus_{j \neq i} k_{ij}\right) \oplus m_i$ but m_i, k_{ij} and b_i have q bits each (not l!). Participant i chooses m_i (his *round* message) randomly, but with exactly p 1's and $q - p$ 0's. The p positions of m_i which contain the bit 1 represent the rounds when participant i is free to broadcast.

All participants can then compute $S = \bigoplus_i b_i$ $(= \bigoplus_i m_i)$, so they all know the rounds in which broadcasts will occur. If no two participants in the network picked the same rounds to broadcast (i.e. the 1's in everyone's round messages occurred in different positions), then the scheme described above functions without any loss of messages. However, this will not usually occur and a certain percentage of messages are always expected to be lost. This can be optimized by choosing the values of p and q well.

Dishonest Participants

In the discussion above, we have assumed that all participants act *honestly*, i.e., they behave in accordance with the protocol. However, a real network may have participants who can prevent the recovery of broadcasted messages by sending random messages when it is not their turn to broadcast. Even one such dishonest participant can ruin the network, so we need a mechanism to detect such participants.

One way to detect dishonest participants in a network is to lay *traps*. To lay a trap, a honest participant chooses some of his broadcasting rounds and decides to broadcast messages that do not need to be kept anonymous. If he then detects that his message has not gone through, he knows that there must be either a collision or a dishonest participant in the network.

If there was no collision, everyone can declare the bits they used to mask their messages and if there is only one dishonest participant, he would be immediately unmasked. If there was a collision, some participant's (perhaps anonymous) message would be revealed which is not desirable. Hence, the protocol should allow a participant to object to such an unmasking. If the same participant objects more than he/she is expected to, he/she is likely to be a dishonest participant.