Dining Cryptographers Network

An anonymous messaging system

Jack Wampler and Ian Martiny

University of Colorado—Boulder

September 27, 2017

Original Problem

Three (or more) cryptographers are eating a meal at a restaurant.

Afterwards the waiter announces that the meal has been paid for

Original Problem

Three (or more) cryptographers are eating a meal at a restaurant.

Afterwards the waiter announces that the meal has been paid for

Only two options exist:

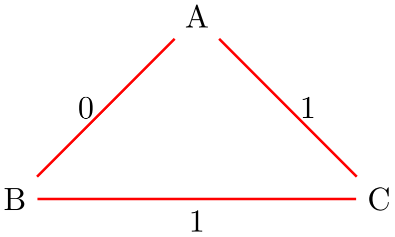
- One of the diners secretly paid for the meal
- The NSA paid for the meal

Original Problem (cont.)

Each cryptographer respects the others' right to pay for dinner but need to know if they did or the NSA paid

- Essentially they need to send an anonymous 1 bit of information

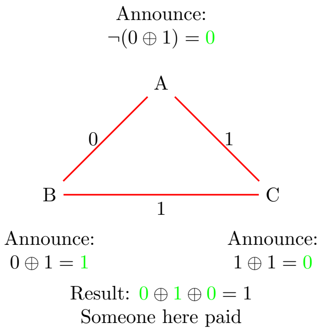
First they need to get a shared bit between each pair (flip a coin behind a menu)



Original Problem (cont.)

Each of the participants follows:

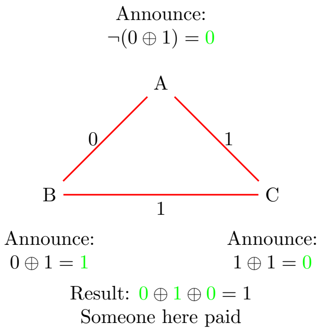
- If they paid: announce the negation of the `xor` of things they know

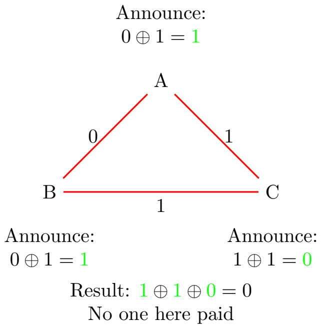


Original Problem (cont.)

Each of the participants follows:

- If they paid: announce the negation of the `xor` of things they know
- If they didn't pay: announce the `xor` of things they know

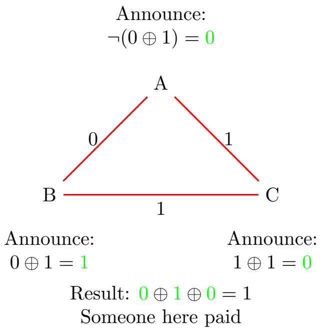


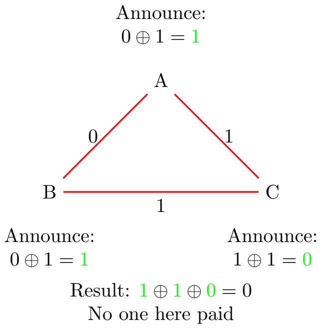


Original Problem (cont.)

Each of the participants follows:

- If they paid: announce the negation of the `xor` of things they know
- If they didn't pay: announce the `xor` of things they know
- `xor` announced data: 0 == NSA paid, 1 == someone paid





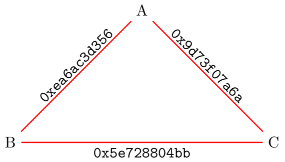
Dining Cryptographers Network

This problem of sending one bit can be extended to sending multiple bits, i.e. a message!
First issue, can't reuse bits! With repeated messages users gain information about who is sending what.

DC Net methodology

1. Establish a shared secret between each pair
   - Diffie Hellman Key Exchange!
2. Use shared keys to seed separate PRNGs
   - Lots of shared secrets!
3. Same protocol as before!

To send “hello”:
`0x68656c6c6f` $\oplus$ `0xea6ac3d356` $\oplus$ `0x9d73f07a6a`
$=$ `0x1f7c5fc553`



In response send:
`0xea6ac3d356` $\oplus$ `0x5e728804bb`
$=$ `0xb4184bd7ed`

In response send:
`0x9d73f07a6a` $\oplus$ `0x5e728804bb`
$=$ `0xc301787ed1`

Result: `0x1f7c5fc553` $\oplus$ `0xb4184bd7ed` $\oplus$ `0xc301787ed1`
`0x68656c6c6f` $=$ “hello”

Design Considerations

Our goal is to be anonymous

- Clients can't just send out messages themselves
- Need a well-behaving server to handle broadcasting messages (playing role of waiter)

It appears there is no more negation in the previous example, but in fact negation is simply the same as `xor`ing by 1. Thus our negate/not negate idea is replaced with an `xor`

Demo