Anonymous Communications

CMSC 414

October 23 & 25, 2017

Why Anonymity?

Does anonymity mean you have something to hide?

What kinds of behaviors does anonymity enable?

- Illegal activities
- Harassment
- Expression of unpopular ideas
- Association with members of marginalized/dissident groups
- Looking up potentially embarassing or stigmatizing information
- Privacy

What is Anonymity

Anonymity refers to the inability of an adversary to determine who is communicating

We can divide this into two main types:

Sender Anonymity Cannot determine the true sender from a set of potential senders

Receiver Anonymity Cannot determine the true receiver from a set of potential receivers

k-**anonymity** means that the adversary cannot distinguish communicant from a pool of k potential parties

Sender Anonymity

Ransom note

Flier on a utility pole

Message in a bottle

Pirate radio station

Receiver Anonymity

Coded classified ads

Number stations

TV/radio receivers (over-the-airwaves, at least...)

Public library

Dining Cryptographers

David Chaum

Given k parties, how can one transmit a message without any of them knowing who it was?

We'll start with 3, and generalize

Alice, Bob, and Carol want to determine who paid for dinner anonymously (one of them or a third party) without violating any of *their* anonymity

Dining Cryptographers

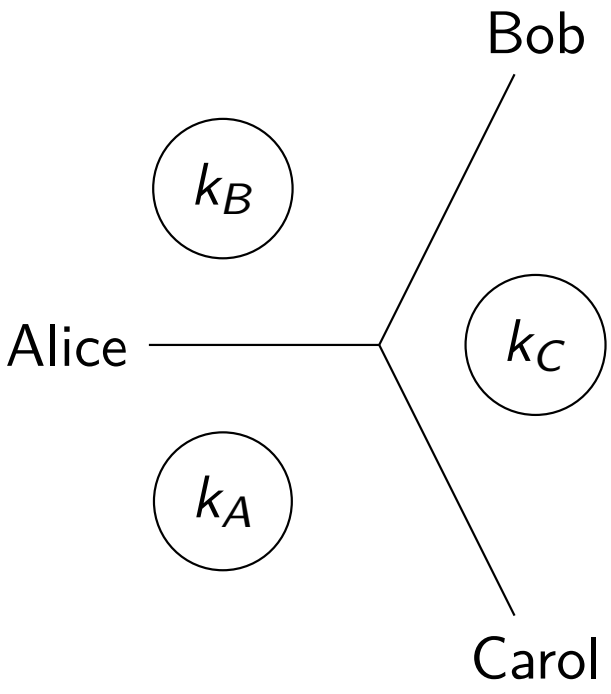
Each flips a coin visible to only two of them

Each then announces XOR of coins they see (heads=1, tails=0)

If Alice paid, she flips her bit

XOR of all announced bits $\Rightarrow$ 1=someone paid

We call this protocol a **dc-net**



Dining Cryptographers

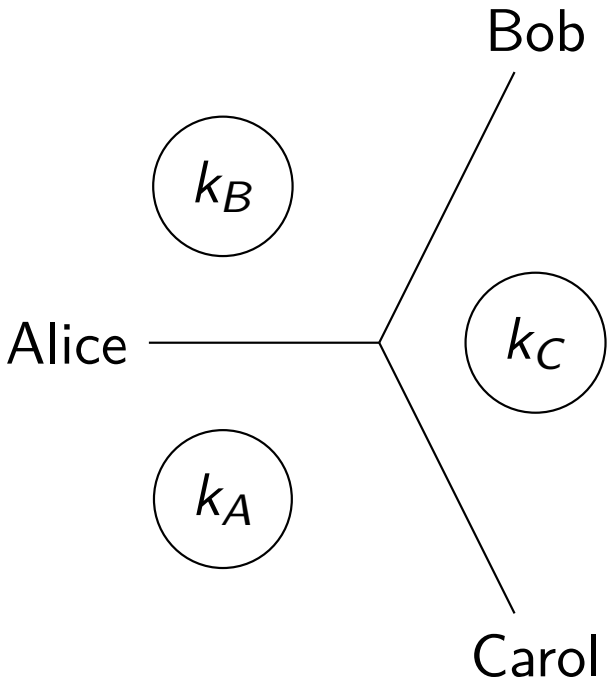
Number the participants

A participant P transmits T_P, where

$$T_P = k_P \oplus k_{P+1} \oplus m_P$$

XOR all these together

$$\begin{aligned} T &= T_A \oplus T_B \oplus T_C \\ &= k_A \oplus k_B \oplus m_A \\ &\quad \oplus k_B \oplus k_C \oplus m_B \\ &\quad \oplus k_A \oplus k_C \oplus m_C \\ &= m_A \oplus m_B \oplus m_C \end{aligned}$$



Generalizing dc-nets

More than 1 bit

$\{k\}$ and $\{m\}$ all must be the same size

The rest of the math holds

This requires a way to generate *sufficiently long* shared keys
$\Rightarrow$ We've seen ways to do this

These keys need to be regenerated for *every message*
$\Rightarrow$ Significant set-up costs

Otherwise, we can eventually distinguish between $m_P = 0$ and $m_P \neq 0$

Generalizing dc-nets

More than 3 participants

Basic extension is simple $\Rightarrow$ pairwise keys between participants

- P_i and P_j share $k_{ij} = k_{ji}$
- $T_i = M_i \oplus \left[\bigoplus_j k_{ij}\right] \forall j \neq i$
- $T = \bigoplus_i T_i = \bigoplus_i M_i$

For n participants, this means $n(n-1)/2$ shared keys to establish for *every message*, and n broadcasts (one from each participant) to send a (possibly null) message

Can make this more efficient, but not without loss of anonymity guarantees

Generalizing dc-nets

More than 1 message

Issues:

1. Continuing communications
2. Collisions

We solve both simultaneously

Scheduled message *time slices* $\Rightarrow$ keys exchanges, bcasts

Any participant P_i sending a message can verify $T \stackrel{?}{=} M_i$
Any participant P_j *not* sending a message has no knowledge of collisions

P_i detects collision $\Rightarrow$ wait a *random number* of time slices before trying again

If messages are *infrequent*, eventually will be sent without collision

Group Exercise 1

You've actually done the hard part of a 3-participant DC-net already. Please clone `anon` with `get-assignment`; task 1 asks you to create a single-round DC-net

Mixnets

Also David Chaum

Given N_S senders, N_R receivers, and a mailserver M, how can we prevent an observer from determining what sender is communicating with what receiver?

By now, should be evident that cryptographers are ~~extremely verbose~~ very precise

We'll use our notation, which assumes encryption and signing are done with proper random padding, etc.

Mixnet Messages

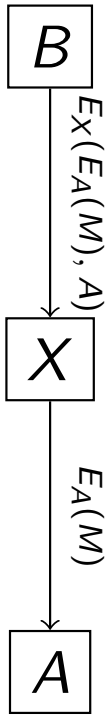
We have a **Mix**, which we will call X

$B \rightarrow X$: $E_X(E_A(M), A)$

$X \rightarrow A$: $E_A(M)$

X cannot actually read messages

On its own, not terribly helpful



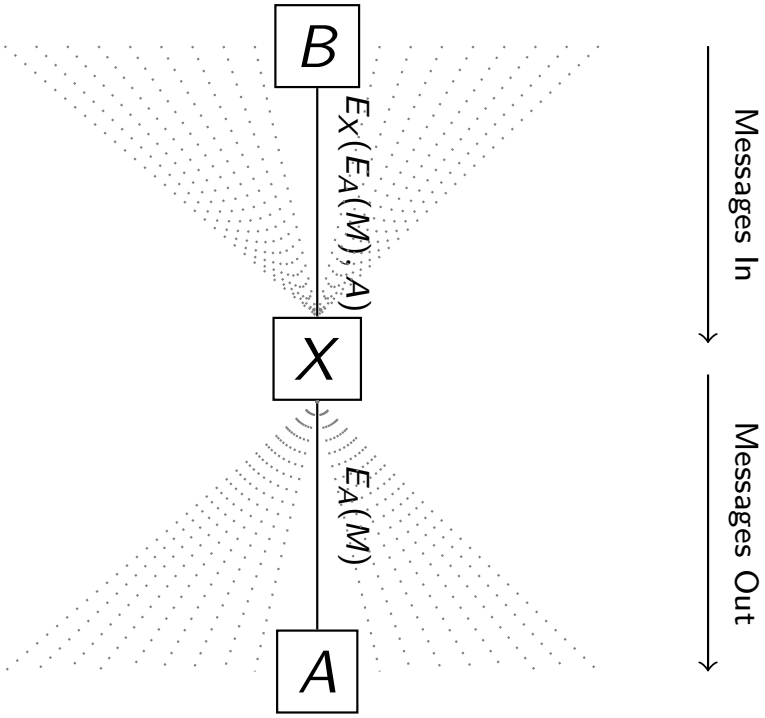
Mixnet Messages

Combine for a lot of senders and receivers

Can see that messages are being sent and received

But *every* potential sender is sending, and *every* potential receiver is receiving

Everything is encrypted
$\Rightarrow$ can't connect inputs/outputs



Mixnet Drawbacks

Mix has to wait for every sender before it can deliver

What if they don't have messages to send?

What if two senders have a message for the *same* receiver?

Receiver can't send a reply to the sender

Mix knows *who* pairs of communicants are, even if not *what* they are communicating

Mixnet Drawbacks

No message from S_i

Similar to what we saw with DC-nets

Senders can encrypt and send garbage

X can throw this out after decrypting

Do this in well-synchronized time slices

Similarly, send encrypted garbage to message-less receivers

Mixnet Drawbacks

Multiple messages for R_i

Much harder to deal with

Cannot send more than one message without revealing something
- $\Rightarrow$ Two senders sent to the same receiver
- $\Rightarrow$ Neither is a dummy message

Mix can *hold* all but one message for R_i until a later time slice

If messages are infrequent, *eventually* all messages will be sent

Message exchange protocol is *synchronous*, but actual message delivery is *asynchronous*

Mixnet Drawbacks

Replies

$\{S\} = \{R\}$

We define a *return address* $E_X(K''_A, A), K'_A$

K'_A and K''_A are *ephemeral* public keys, with $E'_A(x)$ and $E''_A(x)$

$B \rightarrow X$: $E_X(E_A(M, E_X(K''_B, B), K'_B), A)$

$X \rightarrow A$: $E_A(M, E_X(K''_B, B), K'_B)$

$A \rightarrow X$: $E_X(K''_B, B), E'_B(M')$

$X \rightarrow B$: $E''_B(E'_B(M'))$

Note that we need *lots* of padding, but we can make each message repliable and the same length

Mixnet Drawbacks

Matching senders/receivers

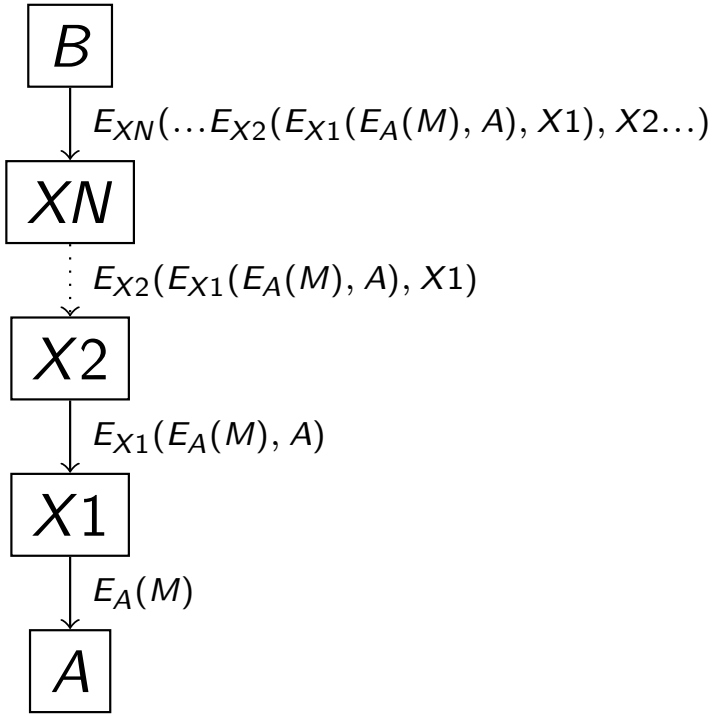
We still have a *single TTP*

Can improve this with a *cascade* of mixes

Have to wrap the encryption in multiple layers

Outermost is first hop from *sender*

Innermost is last hop to *receiver*



Group Exercise 2

In the same repo, task 2 asks you to implement some of the message-passing functionality of a mixnet.

Tor: The Onion Router

Inspired by Mixnets, with slightly different goals:

Deployability Minimal burden (technical/legal) on users, no burden on normal servers

Usability Want as many users as possible

Flexibility Composable with new research and techniques

Simple Design Design/implementation can be validated relatively easily

Provides *low-latency* anonymity, with different security model

How it Works

Alice wants to create a **stream** to Bob

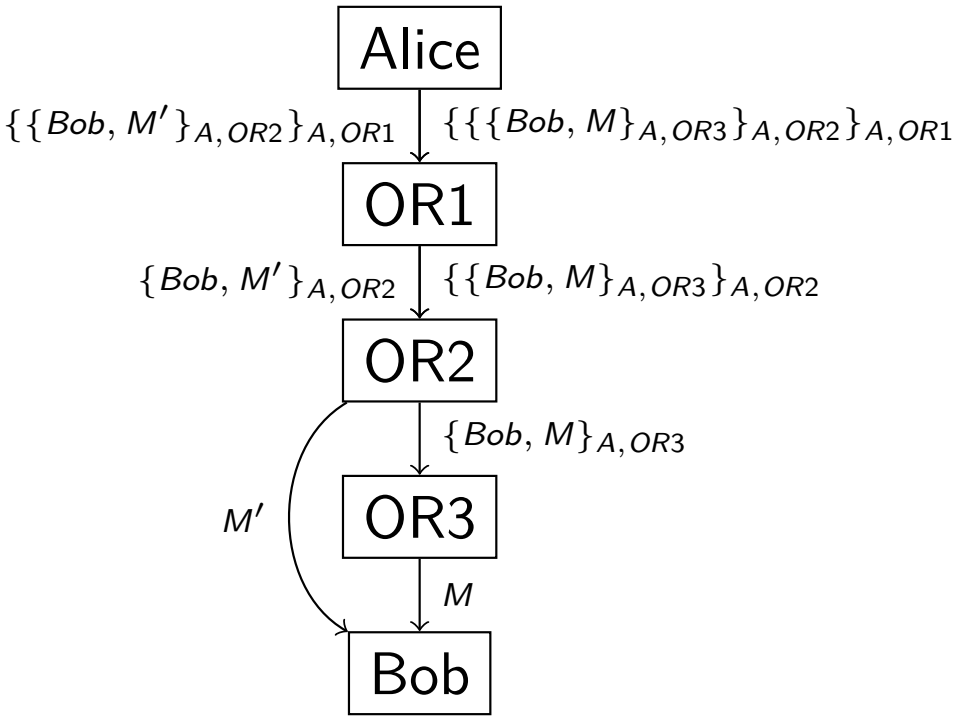
She sets up a **circuit** through Tor

Chooses the relays
- ⇒ How many
- ⇒ Can exit to Bob at any time (**leaky pipe**)

Diffie-Hellman with each relay

Wraps encryptions hop-by-hop

Similar to mixnet cascade



Security Features

All relays maintain TLS connections to one another

Protects against external adversary for confidentiality/integrity

Not against *timing attacks* (correlate messages from Alice/to Bob)

What about malicious insider?

Add integrity checks between Alice and last relay before Bob

Generally sufficient $\Rightarrow$ exit early/re-route to isolate malicious relay

Performance Features

Rate limiting

- enforces long-term average
- allows for short-term bursts

Congestion control

- prevent bottleneck overload
- distinguishes between relay/egress congestion
- both for *circuits* and *streams*

Hidden Services

Sergio wants to run a hidden server

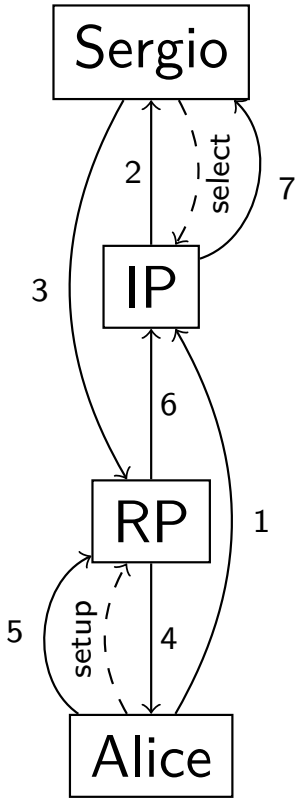
Selects an **Introduction Point** (IP)

Advertises IP w/ K_S

Alice creates a **Rendezvous Point** (RP)

Hands start of D-H handshake to RP, with IP as target

IP relays handshake message to Sergio



Who Knows What?

Alice
: Connecting to a hidden service X, but not who runs it

Sergio
: Running a hidden service X, but not who connects to it

Introduction Point
: Someone is connecting to X, but not who

Rendezvous Point
: Someone is connecting to a hidden service, but not who or what

Comparison with Mixnets

Threat Model

Mixnet assumes a *Global Observer*

- Sees everything
- Passive wiretapper

Tor assumes a *Limited-visibility Observer*

- Sees only some fraction of the network
- Active wiretapper
- Can run their own onion routers (relays)
- Can compromise existing routers

Comparison with Mixnets

Performance/Anonymity Trade-offs

Mixnet batches transmissions

- Cannot correlate senders and receivers based on message times
- Must synchronize based on slowest nodes $\Rightarrow$ high latency
- Makes interactive use (web browsing) difficult; OK for email

Tor transmits immediately

- Can potentially correlate senders and receivers
- Suitable for web browsing & other latency-sensitive apps
- Still adds considerable latency, though
- Relies on enough users to provide *cover* even without batching

Breaking Tor's Anonymity

Given one communicant Alice

$\Rightarrow$ Infeasible to determine who she is talking to

Given *suspected* communicants Alice and Bob

$\Rightarrow$ Can watch both ends and *confirm* based on

- Transmission/delivery times
- Message volume

Attacking Tor

Adversary can control Tor relay nodes

More nodes $\Rightarrow$ more likely to be included in a circuit

DoS other relays $\Rightarrow$ more likely to be included in a circuit

DoS also forces circuits to be rebuilt
$\Rightarrow$ Relays spend more resources on crypto handshakes

Hold exit nodes legally accountable for traffic $\Rightarrow$ discourages use

Exploit misconfiguration and poor choice of options

Group Exercise 3

Task 3 of `anon` asks you to write a *simulator* for a Tor network with an attacker. Such simulators are often useful for analyzing systems by employing what's often referred to as a *toy model*.

Deanonymizing

Even with anonymizing techniques, often possible to **fingerprint** users

Lots of browser/machine-specific information

- Fonts
- Screen dimensions
- Clock skew
- Browser `User-Agent`
- Operating system

Individually, not much information, but

- Combining these narrows potential users considerably
- Requires no client-side tricks (such as cookies)
- Private-mode browsing provides no protection

When Deanonymization is Good

- Detect sources of DoS attacks
- Identify sources of fraud
- Identify account hijackers
- Identify content scrapers

Great, but bad guys can also use this

- No user consent needed (ie, no opt-in)
- No opt-out

Fingerprinting with Fonts

Javascript lets you load fonts

Presence/absence can be sent back to website

Used frequently by analytics companies

Fifield, David, and Serge Egelman. "Fingerprinting web users through font metrics." *International Conference on Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2015.

Boda, Károly, et al. "User tracking on the web via cross-browser fingerprinting." *Nordic Conference on Secure IT Systems*. Springer, Berlin, Heidelberg, 2011.

Fingerprinting with Display Characteristics

Javascript can draw in `<canvas>` element

Can then retrieve pixel-based details, or at least a hash

Will depend on OS, graphics drivers, screen resolution, browser window size, etc

Provides remarkably unique identifier

Mowery, Keaton, and Hovav Shacham. "Pixel perfect: Fingerprinting canvas in HTML5." *Proceedings of W2SP* (2012): 1-12.

Cookies

Third-party cookies can be used to track across websites

Used by advertisers

Advertising services sometimes *share* cookies $\Rightarrow$ **Cookie Synching**

Zombie Cookies come back, even when deleted

Zombie Cookies

Lots of tricks for storing cookies, such as:

- Request to a bogus URL with cookie hash $\Rightarrow$ read web history
- Browser stores HTTP ETags for sites that use them (ostensibly for page freshness)
- HTML5 supports local data storage, independent of cookies
- TCP Fast Open uses a connection-based cookie for reestablishing connections with a server

If a 3rd-party cookie is deleted, any of these can be used by server to put them back

Evercookie is an open-source Javascript package that creates zombie cookies

Zombie cookies can be used on their own, even when 3rd-party cookies blocked

Cross-Device Targeting

Can we identify the *same user* on *different devices*?

Account logins

Similar search or access patterns from the same location

If you have it

- Visual characteristics
  - Facial recognition
  - Lighting/color patterns in environment
- Audio characteristics
  - Voice recognition
  - Environmental sounds
  - *One device emits a sound that the other picks up*

Especially problematic for mobile devices

Countermeaures

To hide in a crowd, have to have a crowd:

- Common OS/browser/settings
- Using Tor puts you in a smaller crowd of a different kind

Can't really disable canvas or fonts

Can block Javascript, but a lot of things will break

Zombie cookies are in a lot of places — Can you find/remove them all? Does your browser let you?

Block 3rd-party cookies, but if zombified before you block them, they might come back!

Group Exercise 4

Look through your browser's stored information. Can you find zombie cookies? Different browsers will store data in different locations. Try deleting some of the cookies from advertising domains, and see if they reappear. If you delete some of the zombie data, do the cookies come back? Does the zombie data come back?

Perhaps more importantly, does your browser provide a way to block these zombie cookies? If so, how difficult does it make it for users to find and understand these settings?

Also, take a look at `http://pantopticlick.eff.org` — what does this tell you about the ability for a 3rd party to fingerprint your browser?