We have thus far investigated how to prove ourselves to (and through) a third party, and how to look up IP addresses of the services with whom we wish to communicate. We will now discuss how to communicate while HIDING our identities from others.

==========

DEFINITIONS OF ANONYMITY

Anonymity is the property that a principal (a user, a piece of software, etc.) has when an attacker is unable to determine with whom it is communicating. We can more concretely define it in terms of what the principal is doing---sending or receiving:

SENDER-ANONYMITY: An attacker overhearing communication cannot determine the true sender from a larger set of potential senders.

RECEIVER-ANONYMITY: An attacker overhearing communication cannot determine the true receiver from a larger set of potential receivers.

These have common analogies in the real world: if a student throws a note while the teacher's back is turned, the teacher may not be able to determine who originated the note. The set of potential senders is the students in the classroom.

For receiver-anonymity, consider FM radio stations: a user may call in and request a song, and dedicate to "You know who"---the set of potential intended recipients is everyone within range of the FM broadcast tower, but no one can discern whom that may be.

These are admittedly toy examples, but we don't have to dig much deeper to arrive at a very real example of receiver-anonymity. It was common during the Cold War for spies to communicate via NUMBER STATIONS (they still persist today). Consider two spies in the same city, and spy A needs to send spy B a message, but without anyone learning who B is---perhaps we even want to keep A from knowing who his fellow spy is! Spy A would make a recording of a (typically automated) voice, saying a coded message (such as "seven, alpha, nine, one, ..."; there are some examples on YouTube). Spy A would then construct a cheap FM transmitter (easy to do with spare parts), and play the message in a loop. Spy B would listen in (with an FM receiver also built using spare parts), and be able to decode the message. But who was the intended recipient? Anyone within broadcast range could have been listening; just like you don't know who may be listening to the radio at any point in time, nor can you determine who may be receiving (let alone capable of decrypting) a message from a number station.

==========

QUANTIFYING ANONYMITY

We often quantify anonymity by the size of the set of potential communicators. If an anonymity protocol does not allow an attacker to be able to differentiate the true sender from a set of N potential senders, then we say that the protocol achieves sender-anonymity of 1/N (likewise for receiver-anonymity).

==========

ACHIEVING SENDER ANONYMITY: DINING CRYPTOGRAPHERS

In the remainder of this document, we will consider some specific protocols that achieve various types of anonymity. Our first is a classic by David Chaum, and the problem is referred to as the DINING CRYPTOGRAPHERS PROBLEM.

- 3 cryptographers (A, B, and C) sit at a dinner table, and the meal has been paid for
- One of them is an NSA agent, but nobody (except the NSA agent himself or herself) knows who it is.
- Those at the table are curious whether the NSA paid for the meal.
- The goal is to derive a protocol by which the NSA agent can broadcast (send to the other two cryptographers) a message bit m, which is either a zero ("the NSA did not pay") or a one ("the NSA did pay"), but without revealing that he is the agent.

Take a moment to consider how you might achieve something like this before reading the PROTOCOL:

(1) Initialization:

Each pair of cryptographers flip a coin while hiding it from the other cryptographer (e.g., by flipping it behind a menu). Let b_AB denote the value of the bit shared by cryptographers A & B, and likewise for the other cryptographers.

(2) Transmission:

Next, each individual cryptographer transmits one of two things. Let's look at it from A's perspective:

- If A is not the NSA agent, then he broadcasts the XOR of the two random bits he shares with B & C: b_AB XOR b_AC

- Otherwise, if A is the NSA agent, then he broadcasts the XOR of the two random bits he shares with B & C, XORed with the message he wishes to send: b_AB XOR b_AC XOR m

(3) Message recovery:

In the final step, each participant XORs together ALL of the messages broadcast in step (2).

Let's go through a concrete example to drive this home.

Initialization: Suppose that we have the following coin flips:

b_AB = 0
b_AC = 1
b_BC = 1

Transmission: Suppose that A is the NSA agent and that he wishes to broadcast a 1. Then here is what each person broadcasts:

A : (b_AB XOR b_AC) XOR m
= (0 XOR 1) XOR 1 = 0

B : (b_AB XOR b_BC)
= (0 XOR 1) = 1

C : (b_AC XOR b_BC)
= (1 XOR 1) = 0

Recovery: Each participant XORs together each of the above broadcast messages to arrive at

0 XOR 1 XOR 0 = 1, the original message.

To see why this is working, let's go through the above example without the 0/1 values. In that case, every participant recovers:

```
      (b_AB XOR b_AC)          XOR m
   XOR (b_AB          XOR b_BC)
   XOR (         b_AC XOR b_BC)
   ================================
                           m
```

To reason about the anonymity of this protocol, let's look at it from the perspective of one of the non-agents: participant B. B knows the bits he shares with A and C (b_AB and b_BC), but not the bit that A and C share (b_AC).

If b_AC = 0, then
* b_AC XOR b_AB = 0 XOR 0 = 0 (what A broadcast)
* b_AC XOR b_BC = 0 XOR 1 = 1 (the opposite of what C broadcast)
* Therefore, C must have been the sender.

If b_AC = 1 then
* b_AC XOR b_AB = 1 XOR 0 = 1 (the opposite of what A broadcast)
* b_AC XOR b_BC = 1 XOR 1 = 0 (what C broadcast)
* Therefore, A must have been the sender.

Because b_AC is chosen by a coin flip, it takes value 0 or 1 both with probability 1/2, and therefore, as far as B can tell, A and C have equal probability of being the sender.

This example demonstrates that A was able to broadcast the same message to all parties without revealing his or her identity.

The above protocol can be extended to support an arbitrary number of participants. The crucial step is to have a shared bit between each pair of participants. The rest of the steps follow as before.

==========

MIX-NETS

We now move to a different scenario, in which N senders (S1,S2,...,SN) each wish to send email to one of N receivers (R1,R2,...,RN). Sitting between the senders and receivers is an email server, M.

Our attack model is as follows: assume an attacker who can view all messages sent to or from the mail server. We refer to this as a GLOBAL, OBSERVER (global because it can see all messages).

The goal of the mail server is to deliver messages in such a way that this global observer is unable to determine which sender is communicating with which receiver.

Here is a PARTIAL BUT FLAWED protocol:

(0) Suppose S_i has message m_i to send to receiver R_i

(1) Each S_i sends R_i||m_i to M, encrypted with M's public key (using hybrid encryption, of course). This indicates "send m_i to R_i".

(2) M BUFFERS each message until it has received a message from each of the N senders.

(3) M then SHUFFLES the messages in a random order. This ensures that

an attacker cannot launch a TIMING ATTACK by correlating the order in which M received messages with the order in which it delivered them.

(4) M sends the messages in this shuffled order, encrypted with the recipient's public key (using hybrid encryption, of course).

This gets us much of the way there, but there are a few problems that we will need to fix.

First, the senders had to trust the mail server with their messages in plaintext. We can address this by having the sender encrypt the message himself. We modify step (1) from above:

(1) Each S_i sends R_i || E(M's public key, E(R_i's public key, m_i)) to the email server.

M decrypts this message to obtain E(R_i's public key, m_i), and thus in step (4), M does not need to re-encrypt the message: he simply forwards this along. Thus, M does not learn the plaintext message, and yet an eavesdropper cannot correlate the messages on either side of M.

The second issue with the above protocol is that it may not be the case that all N senders have anything to send! In that case, the mail server will wait forever in step (2). We maintain anonymity (no one can tell which receiver S_i is communicating with because S_i's messages aren't even reaching any receiver!).

We can address this by having each sender transmit a JUNK MESSAGE to the mail server if it has nothing to send. The mail server treats these messages as if they were real messages. This maintains our anonymity, but at the cost that we are now consuming network resources for otherwise useless messages.

The final issue we wish to address is that we are trusting the mail server with knowing who is communicating with whom. The core idea to solving this is to use a sequence of mail servers:

S -> M_1 -> M_2 -> ... M_k -> D

Suppose we were to somehow construct our messages so that each mail server M_i only learns the mail server that came before it and the one that came after it. Then, assuming that the mail servers are not colluding (working together or sharing information with one another), then:

(1) One of them (M_1) will learn who the source is, but not who the destination is.

(2) One of them (M_k) will learn who the destination is, but not who the source is.

(3) Every other mail server only learns of two mail servers.

These are really great properties! It means that each person on the path learns a little bit of the information, but not enough to determine the precisely who is communicating with whom. All we need to do is construct our messages in such a way that mail servers can forward, without learning anything except the next hop on the path.

To see how we construct such a message, let's start with the message that we know we want D to get:

E(PK_D, msg)

We will need to somehow tell M_k to forward it to D, so what we ultimately want M_k to get is:

```
   E(PK_k, D || E(PK_D, msg))
```

This is the payload we want M_k to get, so we'll need to tell M_{k-1} to send it to him; the message we want M_{k-1} to get must therefore be:

```
   E(PK_{k-1}, M_k || E(PK_k, D || E(PK_D, msg)))
```

This continues all the way to M_1, to whom the source sends:

```
   E(PK_1, M_2 || E(PK_2, M_3 || E(PK_3, M_4 || .... E(PK_D, msg)))...)
```

What we've done is take our original message and add in layer after layer of encryption. Each hop on the path can only "peel back" one layer of encryption, at which point they can only forward it to the next hop (or drop it).

This process is known as ONION ROUTING, and is the basis of Tor ("The Onion Router"). A few more specific details about Tor:

* These end-to-end paths are referred to as CIRCUITS
* Tor (almost) always uses k=3 (3-hop circuits). This balances between anonymity (k is not too small) and latency (k is not too large).
* The last node in a circuit is referred to as an "EXIT NODE": to the outside world, it looks like the exit node is the one initiating connections to destinations (because, after all, it is). Every peer gets to decide whether to be an exit node (or to just relay between other Tor nodes), and for what specific IP addresses/websites they will exit to.
* Clients learn about other Tor peers by downloading a big list of them. You can see a list of active Tor peers---also known as Tor routers---here:

    https://torstatus.blutmagie.de/

You'll notice a few things listed under any give Tor router, including:

* IP address and hostname (so you know how to contact them)
* Country where it resides (so you can decide if you trust it)
* Uptime (so you can decide if it is reliable)
* Average throughput (so you can decide if it's useful)
* What, if any, websites it is willing to be an exit node for
* The node's public key
* ... much more

Tor's attack model differs from mix-nets: in particular, Tor does NOT assume a global, passive attacker. Instead, it assumes that SOME (but far from the majority) of the Tor nodes may be malicious, and that there may be eavesdroppers on small fractions of the links, but without a global view of all traffic.

This is a more "relaxed" attack model compared to mix-nets. What typically comes with assuming a less powerful adversary is the ability to achieve better performance. Here are a couple things that Tor DOES NOT do that mix-nets HAD to do, simply because the attack model is different:

* Tor DOES NOT batch packets
* Tor DOES NOT delay packets

If, for some reason, there were only one client communicating over Tor, then it would have no anonymity (the size of the set of people communicating is 1). So the philosophy behind this relaxed attack model was essentially that, if the performance is reasonable enough, then users will be more likely to adopt it, and the more users who adopt it, the more "cover traffic" there would be -- that is, the harder it would be for an attacker to map any packet to any one sender.

TOR HIDDEN SERVICES

We've discussed how senders can hide their identities. Tor also has a means of allowing destinations to hide their identities. An infamous example of this was The Silk Road: an eBay-like online store where users could purchase illicit and illegal goods for Bitcoins. Running a website like that requires a certain degree of anonymity, and so it was run as what is known as a "Tor Hidden Service".

This website has a fantastic description of this process, with some great diagrams:

https://www.torproject.org/docs/hidden-services.html.en

What's particularly interesting about this process is that it achieves receiver-anonymity by using the techniques that achieve sender-anonymity. Here is a high-level overview:

SETUP:

Suppose we wish to run a service named X on host D without anyone knowing that D is the one running it.

1. D first chooses several "Introduction Points": these are Tor routers who will play an extra role.

2. D constructs a circuit to each of the introduction points it chose and informs them that this circuit should be used for introducing hosts to X.
   (Note that none of the introduction points know who D is.)

3. D constructs an X.onion file and anonymously posts it. This file contains, among other things, a set of introduction points that D has chosen, and the name of the service.

INTRODUCTION:

Suppose a node, S, has obtained the .onion file and wishes to (sender-)anonymously access service X.

4. S first chooses a unique identifier, I, and picks a Tor router to serve as a "Rendezvous Point" R.
   (Note that S does not know who D is.)

5. S constructs a circuit to R, and informs it that if anyone sends it a message for identifier I, to forward that along on this circuit.
   (Note that R does not know who S is.)

6. S then constructs a circuit to one of the introduction points from X.onion, and asks it to forward on the circuit that it has to X a message that contains R's identity and the unique identifier (I) that S chose.

7. The introduction point forwards this on the circuit he has to D.
   (Note that the introduction points still don't know who D is.)

CONNECTION & COMMUNICATION

D, who is running X, has now obtained S's request to "meet up" at the rendezvous point R. S didn't include his own name in the message asking D to meet up, so D does not know who S is.

8. D constructs a circuit to R and uses the unique identifier to inform R that it should "connect" the two circuits together.

9. That is, to send to D, S sends his packets to R, who then forwards them to D, and vice versa. Of course, to ensure confidentiality, S and D should be using end-to-end encryption.

So by the end of this, who knows what?

- D knows that he is the one running X. No one else knows this.
- The introduction point knows that *someone* is accessing X, but does not know that it is S
- R knows that *someone* is accessing *some* hidden service, but does not know that it is S or X.
- S knows that she is accessing X, but does not know who or where X actually is.

In other words, with the above protocol, we obtain sender-anonymity (S is hidden) and receiver-anonymity (D is hidden).

TOR DATA

For more information on Tor, how it is used, and where it is being censored, check out the Tor metrics site:

https://metrics.torproject.org/