Security and Anonymity

- Distributed Systems need a network to send messages.
- Any message you send in a network can be looked at by any router or machine it goes through.
- Further if your machine is on the network receiving message you have to be careful what commands you accept.
Security and Anonymity – Issues

- Encryption — coding messages so that people who look at your data packets can not understand the contents.
- Designing protocols to outwit man in the middle attacks.
- Anonymity — Sending messages so that the sender and/or the receiver can not be identified.
- Secrete sharing — Distributing keys (often in a redundant way) so that a certain number of participants are required open a lock.
Man in the Middle Attacks

The protocol designer expects

Request

Client → Server
Man in the Middle Attacks

The protocol designer expects

Request

Client \rightarrow Server

Reply

Client \leftarrow Server
Man in the Middle Attacks

The protocol designer expects

- **Request**
  - Client → Server

- **Reply**
  - Server → Client

What might happen

- **Client** → **Trojan** → **Server**
Man in the Middle Attacks

The protocol designer expects

![Diagram showing a client communicating with a server, with a Trojan intercepting the communication between them.]

What might happen

![Diagram showing a client communicating directly with a Trojan, which then communicates with the server.]

Request

Client \rightarrow Server

Reply

Server \rightarrow Client

Client \rightarrow Trojan \rightarrow Server
Man in the Middle Attacks

The protocol designer expects

```
Client -> Request <- Server
         |       |
         |       |
         |       |
        Reply
``` 

What might happen

```
Client -> Request <- Trojan
         |       |
         |       |
         |       |
        Reply
```
Man in the Middle Attacks

The protocol designer expects

Client \rightarrow Server

Request

Server \rightarrow Client

Reply

What might happen

Client \rightarrow Trojan \rightarrow Server

Client\rightarrow Trojan \rightarrow Server

Client \rightarrow Trojan \rightarrow Server
Man in the middle attacks

The protocol designer expects

Password?  

Client  

Password=\text{password}  

Server
Man in the middle attacks

The protocol designer expects

Password?

Client ➔ Password=password ➔ Server

What might happen

Password=‽

Client ➔ Password=password ➔ Trojan ➔ Password=‽ ➔ Server

Password=‽

Password=‽

Password=‽
Man in the middle attacks

- Even if things are encrypted it is possible to put make an attack.
- Often protocols work by establishing sessions keys or challenge response
- The Trojan uses copies responses back and forward between the client and the server to open a valid session on the client.
- Designing protocols to avoid all man in the middle attacks can be quite tricky.
Security — Public and private keys

- Encryption algorithms normally work by using public or private keys.
- A private key encryption algorithm is a function $E_k$ which takes a message, $m$,.
Security — Public and private keys

- Encryption algorithms normally work by using public or private keys.
- A private key encryption algorithm is a function $E_k$ which takes a message, $m$, and produces the encoded message $E_k(m)$. 
Security — Public and private keys

- Encryption algorithms normally work by using public or private keys.
- A private key encryption algorithm is a function $E_k$ which takes a message, $m$, and produces the encoded message $E_k(m)$.

- Decoding is done using the same private key: $D_k(E_k(m)) = m$.
Encryption algorithms normally work by using public or private keys.

A private key encryption algorithm is a function $E_k$ which takes a message, $m$, and produces the encoded message $E_k(m)$.

Decoding is done using the same private key:

$$D_k(E_k(m)) = m$$

It is assumed that it is computationally very hard to find $m$ without knowing $k$. 
• Private key algorithms work well when there is a secure way of distributing the keys.
• Two nodes can communicate if they have the same private key. (For example the magic cookies in erlang).
• Public/private keys work by using one key to encrypt and one to decrypt.
Security — Public and private keys

- A node publishes a public key $k_e$.
- If somebody wants to send a message, $m$, it uses the public encryption algorithm:
  \[ E(k_e, m) \]
- The node keys the deception key, $k_d$, secret so to recover the message apply the algorithm:
  \[ D(k_d, E(k_e, m)) = m \]
Security — Public and private keys

- A node publishes a public key $k_e$.
- If somebody wants to send a message, $m$, it uses the public encryption algorithm:
  
  $E(k_e, m)$

- The node keys the deception key, $k_d$, secret so to recover the message apply the algorithm:
  
  $D(k_d, E(k_e, m)) = m$

- The idea is that it easy to encrypt the message but computationally hard to decrypt the message with out the private key.
Security — Implementation

• Normally these algorithms are implemented using various theorems from number theory using prime numbers.

• The fundamental idea is that it is easier to multiply numbers together than it is to factor numbers.

• In general private key algorithms have less computational demands than public/private keys.
The idea is that we have a central server $S$ and two parties $A$ and $B$ which want to communicate with each other:

- $A \rightarrow S$ give me a conversation key to communicate with $B$
- $S \rightarrow A$ retrieve $K_{AB}$ from the message

$$E(K_A, (B, K_{AB}, E(K_B, (K_{AB}, A))))$$

- $A \rightarrow B$ retrieve $K_{AB}$ from the message $E(K_B, (K_{AB}, A))$ that I received from $S$.
- $B \rightarrow A$ (to verify what’s going on and to avoid man in the middle attacks) can you decode $E(K_{AB}, n_B)$ and decrement the argument?
- Here is $E(K_{AB}, n_B - 1)$

Now $A$ and $B$ can communicate without using the authentication server.
"And that he calls for drink, I’ll have prepared for him a chalice for the nonce" (King Claudius to Laertes, Hamlet, Act IV, Scene VII).

- The number $n_B$ generated in the last protocol is a number generated specially for that session and is unique.
- These are used to avoid man in the middle attacks. By sending a encrypted version of it to a known public key ($B$ in this case).
- It is impossible for the man in the middle to send back $n_B - 1$ without knowing the private keys.
Applications of security

- PGP — Pretty good privacy. Send and receive emails using public/private keys.
- SSL — Secure socket layer, encrypted
- Site certificates
- Digital signatures
Onion Routers

- Onion Routers provide a way for two parties to communicate with each other but make it hard for a third party to eavesdrop and find out who is talking to who.
Onion Routers

- Onion Routers provide a way for two parties to communicate with each other but make it hard for a third party to eavesdrop and find out who is talking to who.
- Onion routers need a number of (semi)-trusted nodes, each node has its own (public/private) encryption key.
Onion Routers provide a way for two parties to communicate with each other but make it hard for a third party to eavesdrop and find out who is talking to who.

Onion routers need a number of (semi)-trusted nodes, each node has its own (public/private) encryption key.

If A wants to send a message to B then it picks a route $S_1, \ldots, S_n$ to B.
Onion Routers

- Onion Routers provide a way for two parties to communicate with each other but make it hard for a third party to eavesdrop and find out who is talking to who.
- Onion routers need a number of (semi)-trusted nodes, each node has its own (public/private) encryption key.
- If $A$ wants to send a message to $B$ then it picks a route $S_1, \ldots, S_n$ to $B$.
- It then sends the message:

$$E_{s_1}(E_{s_2}(\cdots E_{s_n}(m + +S_n)) + +S_{n_1})\cdots) + +S_2)$$

to $S_1$. 
Onion Routers

- Onion Routers provide a way for two parties to communicate with each other but make it hard for a third party to eavesdrop and find out who is talking to who.
- Onion routers need a number of (semi)-trusted nodes, each node has its own (public/private) encryption key.
- If A wants to send a message to B then it picks a route $S_1, \ldots, S_n$ to B.
- It then sends the message:

$$E_{s_1}(E_{s_2}(\cdots E_{s_n}(m + S_n)) + S_{n_1}) \cdots) + S_2$$

to $S_1$. 
Onion Routers

- Each router peels off the outer layer and routes it to the next node.
- \( S_1 \) receives

\[
E_{s_1}(E_{s_2}(\cdots E_{s_n}(m + +S_n)) + +S_{n_1}) \cdots ) + +S_2)
\]
Onion Routers

- Each router peels off the outer layer and routes it to the next node.
- $S_1$ receives

$$E_{s_1}(E_{s_2}(\cdots E_{s_n}(m + +S_n)) + +S_{n_1}) \cdots ) + +S_2)$$

- and peels off the outer layer and sends it to $S_2$ as specified in the message.

$$E_{s_2}(\cdots E_{s_n}(m + +S_n) \cdots )$$
Onion Routers

- Each router peels off the outer layer and routes it to the next node.
- $S_1$ receives
  $$E_{s_1}(E_{s_2}(\cdots E_{s_n}(m + +S_n)) ++S_{n_1}) \cdots ) ++S_2)$$

- and peels off the outer layer and sends it to $S_2$ as specified in the message.
  $$E_{s_2}(\cdots E_{s_n}(m + +S_n) \cdots )$$

- Which gets passed to $S_3$ and so on.
Onion Routers

- Onion routers mix it up a bit, extra padding is thrown in to hide messages.
Onion Routers

- Onion routers mix it up a bit, extra padding is thrown in to hide messages.
- To the outside observer it is hard to work out what is routing traffic and what is not.
Onion Routers

- Onion routers mix it up a bit, extra padding is thrown in to hide messages.
- To the outside observer it is hard to work out what is routing traffic and what is not.
- Even if you find a routing packet you’ll need the private keys of all the servers to find out where it is going.
Onion Routers

- Onion routers mix it up a bit, extra padding is thrown in to hide messages.
- To the outside observer it is hard to work out what is routing traffic and what is not.
- Even if you find a routing packet you’ll need the private keys of all the servers to find out where it is going.
- Even if you get hold of the private key of a single router it is no help.
Onion Routers

- Onion routers mix it up a bit, extra padding is thrown in to hide messages.
- To the outside observer it is hard to work out what is routing traffic and what is not.
- Even if you find a routing packet you’ll need the private keys of all the servers to find out where it is going.
- Even if you get hold of the private key of a single router it is no help.
- The more nodes the better, makes it harder to get hold of the private keys.
Onion Routers

- Onion routers mix it up a bit, extra padding is thrown in to hide messages.
- To the outside observer it is hard to work out what is routing traffic and what is not.
- Even if you find a routing packet you’ll need the private keys of all the servers to find out where it is going.
- Even if you get hold of the private key of a single router it is no help.
- The more nodes the better, makes it harder to get hold of the private keys.
- But it does not completely anonymise communication between A and B only makes the routes secure.
A hash function is a function $H$ that takes a message $m$ and produces a hash (most often with less bits).

A good hash function has the property that if $m$ and $m'$ are similar then $H(m)$ and $H(m')$ are very different. You want to randomise up where the data goes into a hash table.

A cryptographic hash is essentially a hash function with a very small chance of collision. That is the probability of the event $m \neq m' \Rightarrow H(m) = H(m')$ happening is very small.

Hash functions are used in digital signature.

In the last lecture they were used in the pastry router.
Freenet

- Freenet is a distributed secure file system.
- Files are identified by a 160-bit SHA-1 hash function (look it up).
- The file system is essentially flat, but it is possible to fake a hierarchical system.
- Users assign descriptions such as `text/philosophy/wittgenstein/investigations`.
- There are also other types of hashes, content-based hash keys which allows files to be split into smaller parts.
Freenet — Retrieving data

- Calculate the hash of what you are looking for.
- Send a message to yourself looking for the file.
Freenet — Retrieving data

- Calculate the hash of what you are looking for.
- Send a message to yourself looking for the file.
- If you don’t find it look up the nearest key in its routing table to the key requested and forward the request to that node.
Freenet — Retrieving data

- Calculate the hash of what you are looking for.
- Send a message to yourself looking for the file.
- If you don’t find it look up the nearest key in its routing table to the key requested and forward the request to that node.
- If the request is successful return the file and cache a local copy and update the routing table.
Freenet — Retrieving data

• Calculate the hash of what you are looking for.
• Send a message to yourself looking for the file.
• If you don’t find it look up the nearest key in its routing table to the key requested and forward the request to that node.
• If the request is successful return the file and cache a local copy and update the routing table.
• For security reasons any node at any time can elect to have the file even if it doesn’t.
Freenet — Retrieving data

- Calculate the hash of what you are looking for.
- Send a message to yourself looking for the file.
- If you don’t find it look up the nearest key in its routing table to the key requested and forward the request to that node.
- If the request is successful return the file and cache a local copy and update the routing table.
- For security reasons any node at any time can elect to have the file even if it doesn’t.
- There is some load balancing as a node caches files it will cache files close to it in the key space.
Freenet — Storing data

- Hops to live — determines the number of nodes that it is stored on.
Freenet — Storing data

- Hops to live — determines the number of nodes that it is stored on.
- Calculate Hash.
Freenet — Storing data

- Hops to live — determines the number of nodes that it is stored on.
- Calculate Hash.
- Look up on local node and nodes in the routing table recursively depth specified by the hops-to-live parameter.
Freenet — Storing data

- Hops to live — determines the number of nodes that it is stored on.
- Calculate Hash.
- Look up on local node and nodes in the routing table recursively depth specified by the hops-to-live parameter.
- If no clash then propagate 'all-clear' message to original insert.
• Hops to live — determines the number of nodes that it is stored on.
• Calculate Hash.
• Look up on local node and nodes in the routing table recursively depth specified by the hops-to-live parameter.
• If no clash then propagate 'all-clear' message to original insert.
• Then propagate the insert message along paths just traversed. At any point a node can declare it self the owner of the file.
Freenet — Storing data

- Advantages
  - Files get propagated to nodes with similar hashes
  - Attacks by trying to insert junk just ends up with the real copies being propagated as well.
• Advantages
  • Files get propagated to nodes with similar hashes
  • Attacks by trying to insert junk just ends up with the real copies being propagated as well.
• Contents of files are normally encrypted.
• Since closeness in the hash-space has no connection with closeness of the files, you have little idea what is on your node.
Dining Cryptographers

Three cryptographers at a dinner.

Either the NSA has paid or one of the cryptographers have paid. The cryptographers respect each other right to privacy.
Dining Cryptographers

1. Each cryptographer tosses a coin (H or T) and shares it with the cryptographer on his left.
2. If the coins are different then announce that it is odd.
3. In normal operation there should be an even number of differences 0 or 2.
Dining Cryptographers
Dining Cryptographers
Dining Cryptographers

\begin{center}
\begin{tikzpicture}
\node[draw,shape=circle,thick,minimum size=1em] (H1) at (0,0) {$H$};
\node[draw,shape=circle,thick,minimum size=1em] (H2) at (1.5,0) {$H$};
\node[draw,shape=circle,thick,minimum size=1em] (H3) at (3,0) {$H$};
\node[draw,shape=circle,thick,minimum size=1em] (T1) at (4.5,0) {$T$};
\node[draw,shape=circle,thick,minimum size=1em] (T2) at (6,0) {$T$};
\node[draw,shape=circle,thick,minimum size=1em] (H4) at (0,-1.5) {$H$};
\node[draw,shape=circle,thick,minimum size=1em] (T3) at (4.5,-1.5) {$T$};
\draw[->,thick] (H1) -- (H2) node [midway, above] {$S$};
\draw[->,thick] (H2) -- (H3) node [midway, above] {$S$};
\draw[->,thick] (H3) -- (H1) node [midway, below] {$S$};
\draw[->,thick] (T1) -- (T2) node [midway, above] {$D$};
\draw[->,thick] (T2) -- (T3) node [midway, above] {$D$};
\draw[->,thick] (T3) -- (T1) node [midway, below] {$D$};
\end{tikzpicture}
\end{center}
Dining Cryptographers

- If a cryptographer wants to broadcast a signal anonymously, then he just lies about the two coins being same or different.
- Hence there will be an odd number of differences.
- The other cryptographers know somebody has signaled but there is no way of knowing who.
For example if the top cryptographer wants to pay:
Dining Cryptographers

For example if the top cryptographer wants to pay:
Dining Cryptographers

For example if the top cryptographer wants to pay:

- D → H → S
- S → T → D
- S → T → D
Dining Cryptographers

For example if the top cryptographer wants to pay:

Generalises to an odd number of cryptographer, also an odd number of people each to send a message without knowing which message came from which person.
• There are other protocols for various types of anonymous voting.
• The problem has a general statement: Compute

\[ F(x_1, x_2, \ldots, x_n) \]

without revealing any individual \( x_i \).