Just one brief announcement.

HKN reviews are going to be done in class next Monday so you guys should make sure you come, give us your feedback, let us know what you like and what you don't like.

And, with that, we'll start talking about protection.

Protection is like fault-tolerance and recoverability.

One of these properties of systems, or building secure and protected systems has implications for the entire design of the system.

And it's going to be sort of a set of cross-cutting issues that is going to affect the way that, for example, the networking protocols are designed or that the sort of modules that make up your biggest computer system are designs.

So it's going to be a whole set of usually that we're going to look at through the course of this discussion about protection that are going to affect the system at all levels.

In 6.033, we use the work protection and security essentially synonymous.

Often times we'll talk about a secure system or a system that has security or certain security goals that we have, so we're going to use those words interchangeably.

Security is one of these topicals that you guys are familiar with to some extent already.

You've heard about various things on the Internet going on where people's information has been stolen on laptops or a website has been cracked into or some worm or new virus, the new I love you virus is spreading around and confection people's computers.

So you guys are sort of familiar with it on a collegial sort of way, I'm sure.

You also are familiar with many of the tools that we're going to talk about, so the applied versions of the many of the tools that we're going to talk about.

You've all used a password to lock into a computer before or you've used a website that using SSL to encrypt the communication with some other website.

So you're going to be familiar with some of many of the high letter instances of the tools that we'll talk through in this session, but what we're going to try to delve down into in 6.033 is how those systems are actually put together, what the design principals are behind building these secure systems.

As I said, you guys are presumably very familiar with, you've heard about these various kinds of attacks that are going on.

And one of the things that's happened in the last few years, as the Internet has become more and more commercial and larger and larger, is that it's meant that security of computers has become much, much more of a problem.

So the growth of the Internet has spawned additional attacks.

If you go look at a website, for example, there are several security websites that track recent security breaches This is one example.

It's called Security Focus dot com.

I don't know if you can see these, but this is just a list of things that have happened in the last just few days on the Internet.

Somebody is reporting that web server hacks are up by one-third, some IT conference was hacked, Windows says "trusted Windows" is still coming.

[LAUGHTER] So it just goes on and on with these are things that have happened in the last few days.

There is this huge number of things.

You may have heard recently about how there have been several large companies recently that have had big privacy problems where databases of customer information have been stolen.

AmeriTrade just had this happen.

The University of California at Berkeley had something like several hundred thousand applications and graduate student records were on a laptop that was stolen.

These are the kinds of things, the kinds of attacks that happen in the world, and these are the kinds of things that we're going to talk about how you mitigate.

The objective, really, of security, one simple way to look at an objective of security is that we want to sort of protect our computer from bad guys.

The definition of bad guy depends on what you mean.

It could be the 16 year old kid in his dorm room hacking into people's computers.

It could be somebody out to sleep hundreds of thousands of dollars from a corporation, but let's assume that there are some bad people out there who want to sort of take over your computer.

But, at the same time, the objective of security is also to allow access to the good guys.

So one way to make sure that the bad guys don't get at your data is simply to turn your computer off, right?

But that's not really a good option.

We want the data to be available to the people who need the data and have the rights to access the data.

Often times we can sort of frame our discussion, we can say that we're trying to protect we have some set of information that we want to keep private.

So sort of a goal of a secure system, in some sense, is providing privacy.

So we have some set of data that's on our computer system or that's being transmitted over the network that we want to keep private, we want to keep other people from being able to have access to or tamper with.

And throughout the sort of notion of what it means for a computer system to be secure is sort of application dependent.

It depends very much on what the computer system we're talking about is.

It may be the case that in your file system, you have a set of files that you want the entire world to have access to, stuff that's on your webpage they're willing to make public.

You don't have any real security concerns about who can read it.

But, at the same time, you might have banking data that you really don't want people in the outside world to be able to access.

So you have a set of policies that define, in your head, some sort of set of policies or rules about what it is that you would like users to be able to access.

So almost any system has some policies associated with data should be accessed by other people.

Some notion of what data they want to keep private that can be sort of translated into this set of policies.

So in 6.033 it's sort of hard to study in a systematic way different types of policy.

Policy is something that we're not going to have, we could sit around and have an informal discussion about different possible policies that you might want.

But instead, in 6.033, what we're going to do, as we've done in much of the rest of the class is talk about mechanisms that we can use to enforce these different security policies that someone might have.

So we're going to talk about the tools that we use to protect data.

In thinking about security and in thinking about these mechanisms, it's useful to start off maybe by thinking about what we mean by security and protection in the real world and sort of compare the mechanisms that we have in the real world for protecting data from, say, mechanisms that we have on the computer.

From the point of view of what we might want to accomplish with a secure computer system, some of the goals and objectives are similar to what we have in the real world.

Clearly, we have this same objective which is to protect data.

We can say, just like in the real world, we have a lock on a door that protects somebody from getting access to something we don't want them to have access to.

In the world of computers, we can encrypt data in order to make so somebody who we don't want to have access to our data doesn't have access to that data.

Similarly, there are also, say, for example, in the real world a set of laws that regulate who can access what data and what's legal and what's not legal.

It's not legal for me to break into your house and take something from it.

Similarly, it's not legal for somebody to hack into a computer and steel a bunch of files.

There are also some differences.

The obvious one is one that has been true of almost all of our comparisons between the real world and, say, normal engineering disciplines that involve building bridges and buildings and computer systems.

And that's this issue that computer systems have a very high dtech over dt.

So the computer systems change quickly.

And that means there is always both new ways in which computers systems are connected to the world, there are new and faster computers that are capable of breaking encryption algorithms that maybe we didn't think could be broken before, there are new algorithms being developed both for protecting data, and there are new strategies that people are adopting to sort of attack computers.

In the recent years we've seen this thing where people are doing what they call phishing, P-H-I-S-H-I-N-G. Where people are putting up fake websites that look like real websites.

So all these emails that you get from Bank One or whoever it is saying come to our website, click on it and give us your social security number and your credit card number, I hope you guys aren't responding to those.

Those are fake websites and people are trying to steal your information.

We see new attacks emerging over time.

The other kinds of things that we see that are different, clearly computer systems are, a tax in computer systems can be both very fast and they can be cheap.

So, unlike in the real world, in a computer system you don't have to physically break into something.

You can do this very quickly over a computer system.

So you see, for example, with some of these warms and viruses, these things spread literally across tens of thousands of computers in a matter of seconds.

So these are very efficient and effective attacks that can take over a huge number of computers in a very short period of time.

And that leads to us wanting to have a different set of sort of mechanisms for dealing with these kinds of problems.

And then, finally, it is also the case there are some differences in between laws and computer systems and in the real world.

In particular, because the computer systems change so fast because new technologies develop so fast, the legal system tends to lag significantly behind the state of the art and technology.

So the legal system often doesn't have regulations or rules that specifically govern whether or not something is

OK or is not OK.

And that means sometimes it's unclear whether it's legal to do something.

So, for example, right now it's not clear whether it's legal for you to take your laptop out in the City of Cambridge, open it up and try and connect to somebody's open wireless network.

Certainly, you can do that, it's very easy to do, probably many of us have done this, but from a legal standpoint there is still some debate as to whether this is OK or not.

What this suggests, the fact that laws are often unclear, ambiguous or simply unspecified about a particular thing is that we're going to need additional sort of sets of, if you really want to make sure your data is secure, if you want to enforce a particular security policy you're going to need to rely more on sort of real mechanisms in the computer software to do this rather than on the legal system to say, for example, protect you from something that might be happening in the outside world.

Designing computer systems is hard.

A secure computer system is hard, in particular.

And the reason for that is that security, often times the things we want to do in secure systems, the things we want to enforce are so-called negative goals.

So what do I mean by that?

An example of a positive goal is, for example, I might say Sam can access pile F.

That, presumably, is something that is relatively easy to verify that that's true.

I can log onto my computer system.

And if I can access this file F then, well, great, I can access the file F.

We know that's true.

And that was easy to check.

Furthermore, if I cannot access the file and I think that I should have the rights to access it, I'm going to email my system administrator and say, hey, I think I should be able to access this file, why can't I, will you please give me access to it?

An example of a negative goal is that Sam shouldn't be able to access F.

At first it may seem that this is just the same problem as saying, it's just the inverse of saying Sam cannot access F, but it seems like it should be just as easy or hard to verify.

But it turns out that when you're thinking about computer systems, this sort of a problem is very hard to verify because, while it may be true that when I try and open the file and I log into my machine and I connect to some remote machine and try and access that file, it may be the case that the file system denies me access to that file.

But I may have many other avenues for obtaining access to that file.

For example, suppose I have a key to the room in which the server that hosts that file is stored.

I can walk into that room and may be able to sit down in front of the consol on this machine and obtain super user route access to that machine.

Or, I may be able to pull the hard drive off the machine and put it into my computer and read files off of it.

Or, I may be able to bribe my system administrator and give him a hundred dollars in exchange for him letting me have access to this file.

So there are lots and lots of ways in which users can get unauthorized or unintended access to files or other information in computer systems.

And verifying that none of those avenues are available to a particular user is very hard.

Worse, or similarly, this is hard because it's very unlikely that a user is going to complain about having access to some file that they shouldn't have access to.

I'm not going to call up my system administrator and say, hey, I have access to this file, I don't think I should have access to it.

Nobody is going to do that.

So, even though I'm not a malicious user, I don't really have any incentive to go to my system administrator and tell them that there's this problem with the way that things are configured on the computer.

And that extends also to people who, in fact, are malicious users.

When somebody breaks into a computer system, they don't typically, usually, send out an advertisement to everybody in the world saying, hey, by the way, I got access to this file that I should have access to.

It's possible for them to log in, read the file, log out, and nobody would be any of the wiser.

Many of our security goals are negative, and that means building secure computer systems is hard.

What we're going to do in 6.033, in order to get at this, get at sort of building secure systems in the face of these negative goals, is look at a set of different security functions that we can use to protect information and access to computers in different sorts of ways.

And we're typically going to talk about, throughout this, a client server sort of a system where you have some client and some server that are separated, typically over the Internet, that are sort of trying to exchange information or obtain information from each other in a way such that that information exchange is protected and secure and so on.

Suppose we have a client sending some information out over the Internet to some server.

What are the kinds of things that we want to make sure, what are the sorts of security goals that we might want to enforce in this environment?

One thing we might want to do is authenticate the client.

The server might like to know for sure that the person who issued this request is, in fact, the client.

They'd like to have some way of knowing that the client issued this request and that the request that was sent is, in fact, what the client intended to be sent.

For example, that somebody didn't intercept this message as it was being transmitted over the network, change it a little bit and then send it on making it look as though it came from the client.

There might be different kinds of attackers that are sitting in this Internet.

For example, there might be say an intermediate router along the path between the client and the server where the system administrator of the router is malicious.

Oftentimes, in the security literature, there are these funny names that are attached to the different people in different places.

Oftentimes the client is called Alice and the server is called Bob, the person receiving the request.

And we talk about two different attackers.

We talk about Eve who is an eaves dropper who listens to what's going on and tries to acquire information that she's not authorized to have and we talk about Lucifer who is the bad guy who not only is trying to overhear information but may do arbitrarily bad things.

He's trying to take over the data and corrupt it in any way he possibly can.

One goal is that we want to prevent, say, for example, Lucifer from being able to interfere with packets coming from Alice to Bob.

We want to make sure that packets the server receives actually originated from Alice and were the original request that Alice sent, so that's authentication.

We also might want to authorize, at the server, that Alice is, in fact, allowed to access the things that she's trying to access.

If Alice tries to read a file, we need some way of understanding whether Alice is allowed to access this file or not.

We also need to keep some information confidential.

We may want it to be the case that Eve, who overhears a packet, cannot tell what the contents of that packet are.

We might want to be able to protect the contents of that thing so that Eve cannot even see what's going over the network.

So notice that this is a little bit different than authenticating.

Authenticating says we just want to make sure the packet, in fact, came from the client.

But we're not saying anything about whether or not somebody else can see the contents of that packet.

Keeping confidential says we want to make sure that nobody, except for the intended recipients can, in fact, see the contents of this.

There are a couple other properties that we want as well.

One thing we might want is accountability.

We're going to talk about this a little bit more.

This says we need to assume that it's always possible that something could go wrong.

It's always possible that I might have bribed my assistant administrator and he might have given me access to the computer.

And, in the end, there is not much you're going to be able to do about that.

What you want to do is make sure that when situations like that occur that there's some log of what happened, you have some way of understanding what it was that happened, why it happened and how it happened so you can try and prevent it later.

You want to make sure you do a counting, you keep track of what's been going on.

And, finally, you might want availability.

This is you might want to make sure that Lucifer, who is sitting here between the client and the server cannot, for example, send a huge number of packets at the server and make it unavailable, just swamp it with a denial of service attack.

Availability means that this system, in fact, functions and provides the functionality that it was intended to provide to the client.

We are going to spend a while in 6.033 especially focusing on these first three techniques.

Essentially, the next three lectures are going to be talking about how we guaranty, how we authenticate and authorize users and how we keep information confidential and private.

But all of these goals together there is a set of technical techniques that we can talk about for trying to provide each one of these things.

But when you think about building a secure system, it's not enough to simply say we're going to employ, you know, I employ authentication to make sure that Alice is, in fact, Alice when she talks to Bob.

What you want to do, when you build a secure system, is think about sort of how to, you want to get your mindset around building sort of the general ideas behind a secure system.

And so, in 6.033, we have this set of principles that we advocate called the safety net approach.

And the idea is the safety net approach is a way to help you sort of in general think about building a secure system as opposed to these specific techniques that we're going to see how to apply later on.

The safety net approach advocates sort of a set of ways of thinking about your system.

The first one is be paranoid.

This is sort of the Murphy's Law of security.

It says assume that anything that can go wrong will go wrong.

Don't just assume that because you're authenticating the communication between Alice and Bob that there is no way that somebody else will pretend to be Alice.

You should always have something, a safety net, some backup to make sure that your system is really secure.

A good example of being paranoid and applying the safety net approach are things like suppose your router has a firewall on it, suppose your home router has a firewall that's supposed to prevent unauthorized users from being able to access your computer, does that mean that you should then turn off all password protection on your computer so that anybody can log in?

No, you're not going to do that.

You're going to continue to protect the information on your computer with the password because you may have a laptop and may not be using it from home.

Or, you may be worried about somebody breaking into your computer from inside of your house, say, perhaps.

That's an example of sort of thinking about the safety net approach.

You have multiple layers of protection for your information.

There are some sort of sub-approaches, there are some sort of sub-techniques that we can talk about in the context of being prepared.

One of them is accept feedback from users.

If you were designing a big computer system and somebody tells you that something is secure or that there is a problem, be prepared to accept that feedback, have a way to accept that feedback and respond to that feedback.

Don't simply say oh, that's not a real security problem, that's not a concern or don't, for example, make it so the users don't have a way to give you information.

Defend in depth.

This is have multiple security interfaces like passwords plus a firewall.

And, finally, minimize what is trusted.

You want to make sure that the, and this is sort of a good example, we've talked about this as an example of system design before.

We want to try and keep things as simple as possible, but this was really important in the context of security because you want to make sure, for example, that you try and keep the protocols that you use to interact with people from the outside world as simple as possible.

The more interfaces that you have with the outside world that users can connect to your computer over, the more places you have where your computer system is vulnerable.

So you want to try and minimize, make it so that your computer system has as few sort of openings it can that you have to verify our securer as possible.

Other examples of the safety net approach, you need to consider the environment.

This just means it's not enough, you know, suppose that I have this connection here, this connection over the Internet between Alice and Bob, I may assume that I'm only worried about attackers who are coming in, say, for example, over the Internet.

But if you're a server, the server may have other connections available to it.

So it may be the case that this is a server inside of some corporate environment and this server has a dialup modem connected to it.

Especially, nowadays, this is less common, but it used to be the case that almost all companies had a way that you could dial in and get access to a computer when you didn't have a wired Internet connection available to you.

And often times this dial-in access was a separate place where people could connect.

So, for example, it didn't have the same interface as the sort of main connection to the Internet.

And that meant that there was sort of side channel, in the environment, through which people could use to get access to the computer.

Similarly, this means you should think about all the people who have access to the computer.

Is it the case maybe that this is a computer in your office and the janitor who works in your office comes into the office every night, would he or she be able to sit down in front of your computer and get access to the system when they shouldn't be able to?

And this may sound paranoid.

It is being paranoid, but if you really want to build a secure computer system you need to sort of keep all these things in mind.

Need to plan for iteration.

This is, again, just a good system design principle, but it's especially true here.

Assume that there will be security violations in your computer system, assume there will be security problems, and plan to be able to address those problems and also have a way to verify that once you've addressed those problems the other parts of your system that are supposed to be secure continue to be secure.

And, finally, keep audit trails.

This gets at our goal of accountability.

This just means keep track of everything.

All of the authentication and authorization requests that you made in your system, when a user logs in, keep track of where they logged into, maybe even keep track of what they did so that you can come back, if it turns out that this person was unauthorized, you later discovered that they were up to no good, you can come back and understand what it is that they did.

What all this sort of discussion, especially this discussion about the safety net approach illustrates or gets at is that there are lots of these issues that we're talking about.

So, for example, the janitor breaking into our computer or me bribing my system administrator.

There aren't really computer system issues, right?

These are human issues.

These are things that I'm bypassing any sort of authentication I might have in the computer or any kind of security that I might have built into the computer because, for example, my system administrator is authorized to access anything on the computer he wants to.

And so I have, essentially from the computer systems point of view, made it look like I have rights to get at anything I want to if I can bribe the system administrator.

So this suggests that sort of in many computer systems humans are the weak link.

Not only are people bribable but people make mistakes.

People don't read dialog boxes.

People do things hastily.

People don't pay attention.

The reason that these phishing attacks work where somebody pretends to be Washington Mutual or eBay and sends you an account that asks you to log in and type in your social security number is that people don't think.

They just see this email and say oh, I guess eBay wants me to give them my social security number.

OK.

People make mistakes.

And this is the way that many, many security vulnerabilities or security problems happen is through people mistakes.

So we're going to talk in most of 6.033 about technical solutions to security problems.

But, when you're actually out in the world building a system, you need to be thinking about the people who are going to be using this system almost as much as you're thinking about the sort of security cryptographic protocols that you design.

That means, for example, you should think about the user interface in your system.

It does the user interface to promote sort of users thinking securely.

The classic example of sort of a bad user interface is your web-browser popping up this dialogue box every time you access a site that isn't SSL encrypted saying this website is not SSL encrypted, are you sure you wish to continue?

And, after it has done this about ten times and these are sites that you know and believe are safe like almost any site on the Internet, you just click the dialogue box that says never show me this alert again.

There is almost no useful information that the system is giving you by complaining in that way.

Other examples of things you want to make sure you do is have good defaults.

In particular, don't default to a state where the system is open.

When an error occurs, don't leave the system in some state where anybody can have access to anything that they want to.

Instead, default to a state where people don't have access to things so that you're not exposing the system to failures.

Don't default to a password that anybody can guess.

When you create a new user for your system don't make the default password PASSWORD, that's a bad idea.

Make it some random string that you email to the user so that they have to come back and type it in.

Finally, give users least privilege needed to do whatever it is they need to do.

This means don't, by default, make users administrators of the system.

If the user doesn't need to be an administrator of the system, they shouldn't have administrative access, they shouldn't be able to change anything that they want.

A good example of least privilege being violated is many versions of Microsoft Windows, by default, make the first user who is created an administrator user who has access to everything.

And this is a problem because now when somebody breaks into that users account they now have access to everything on the machine, as opposed to simply having access to just that user's files.

In general, what this just means is keep your systems simple, keep them understandable, keep the complexity down.

And that's sort of the safety net.

There are these two principles that you want to think about.

One, sort of be paranoid.

Apply this notion of having a safety net in a computer system.

And, two, don't just think about the technical protocols that you're going to use to enforce access to the computer but think about sort of who is using this system, think about humans who have access to it, and think about how to prevent those humans from being able to do stupid things that break your security goals for your system.

This was a very sort of high-level fuzzy discussion about security.

Now what we're going to do is we're going to drill in on some of these more specific technical protocols.

And today we're going to look at a way in which you can think of most secure systems as consisting of a set of layers, and those layers are basically as follows.

We have, at the top, some application which we want to secure.

And then underneath this application we can talk about three layers.

So we have some kind of functionality that we want to provide, we have some set of primitives that we're going to use to provide that, and then, at the very bottom, we have cryptography which is the set of mathematics and algorithms that we're going to use that we generally, in modern computer systems, use to make sure that the computer system is secure.

The application may have its high-level functions that correspond to these things over here, so we may want to be able to authenticate users or we may want to be able to authorize that users have access to something or we may want to provide confidentiality.

So we may want to be able to make sure that nobody can read information they don't have access to.

So you're going to have a set of primitives for doing these different things.

For authentication, we're going to need to talk about something called an access control list.

For authorization, we're going to talk about primitives called sign and verify.

And for confidentiality we will talk about these primitives called encrypt and decrypt.

And these topics, this stuff that's in this middle set of primitives, we're going to describe these in more detail in later lectures.

What I want to do with the rest of the lecture today is to talk about this bottom layer, this cryptography layer.

We have some set of cryptographic ciphers and hashes.

And so a cipher is just something that takes in, say, a message that the user wants to send that is not protected at all.

And it flips the bytes in that message around in order to create something that is not understandable unless you have some piece of information that allows you to decipher that message.

You can say ciphering and deciphering is sort of like encrypting and decrypting, words you may be familiar with.

We also talk about hashes.

Hashes we're going to use to authenticate or to authorize a particular message to make sure that a message is -- I'm sorry.

Just for your notes, I got this backwards.

This should be authenticate with sign and verify and we authorize with ACL.

We'll talk about these things more in a minute, but it was just confusion over two words that start with auth.

So we're going to use hashes to basically authenticate that a user is, in fact, who they claimed that they were.

And, again, we'll see in more detail how this works over the next couple days.

Early cryptographic systems relied on this idea, or early cryptography relied on this idea that we're going to try and keep the protocol that's used for encoding the information secret.

A simple example of an early encryption method might be something that many of you played with when you were a kid where you transpose all the letters.

You have some map where you A maps to C and B maps to F and so on, some mapping like that, and you use that to encrypt it.

And there are these puzzles where you get told a few of the letters and you try and guess what the other letters and decode a message.

That's a simple example of a kind of encryption that you might apply.

And that encryption relies on the fact that this transform is essentially secret.

If you know the transform obviously you can decrypt the message.

These schemes are often called closed design crypto schemes.

And the idea is that because the attacker doesn't know what scheme was used to encode the information it can be very, very hard for them to go about decoding it.

For example, the architecture for this might look like message goes into some encryption box which is secured from the outside world.

It was just hidden from the outside world.

Nobody else knows what that is.

And this goes over the Internet to some other decryption box which then comes out on the other end as a message.

This is a closed design.

And these designs, in general, sort of turn out not to be a very good idea because the problem is if somebody does discover what this function is now you're in trouble.

Now this whole system is no longer secure.

And, worse than that, when you make these things secure, if you suppose now you're going to put this system out in the world with a hidden protocol that nobody knows in the world, now it's sort of you against the whole world.

Whereas, if you had told everybody what the protocol was to begin with and said this is the protocol and there is this little bit of information that we keep secret within the protocol, the two parties in the protocol keep secret, but here's the sort of algorithm that's in the protocol, let's let the entire world verify whether this protocol is, in fact, secure or not, you'd have a much better chance of developing something that was secure.

It is sort of accepted wisdom that these kinds of closed design systems tend not to be widely used anymore.

The systems that we are going to talk about, for the rest of this talk today, are going to be so-called open design systems.

Of course, there are many, many systems that were designed with this and there are many, many closed cryptography systems.

And, in some ways, they're sort of the most natural ones and the things you'd think of first.

And they've been very effective throughout history.

It's just the case that modern cryptography typically doesn't rely on it.

Open design systems have an architecture that typically looks as follows.

It is pretty similar to what we showed before.

We have m going into E.

But this time this protocol E, the world knows what the algorithm E is and instead this E has some piece of secret information coming into it, a key, k.

And this key is usually not known to the world.

And now we go through the Internet, for example, and we come out to a decryption box which also has a key going into it.

And these keys may or may not be the same on the two boxes.

And we'll talk about the difference between making them the same or not making them the same.

Now we have this message that comes out.

Message comes in, gets encrypted, goes over the Internet, goes into the decryption box and gets decrypted.

If k1 is equal to k2 we say this system is a shared secret system.

And if k1 is not equal to k2 we say that this is a public key system.

In k1 is equal to k2 these two keys are the same.

And, say, Alice and Bob on the two ends of this have exchanged information about what this key is before the protocol started.

Alice called up Bob on the phone or saw Bob in the hallway and said hey, the key is X.

And they agreed on this beforehand.

And now that they've agreed on what this key is they can exchange information.

In a public key system, we will see the design of how one public key system works in a little bit more detail, but typically it's the case that, for example, the person who is sending the message has a private key that nobody else knows, only they know, and then there's a public key that everybody else in the world knows and is sort of distributed publicly.

And these two keys are not equal but there is some mathematical operation that you can apply that, given something that's been encrypted with k1, you can later decrypt it with k2. And we will look at one example of one of those mathematical functions.

The point here is closed design says the algorithm itself is unknown to the world.

What shared secret simply says is that there is some little bit of information, a little key, like a number that we've exchanged, but it's not the algorithm, it's not the protocol, it's just this one little bit of, say, several hundred bits of key information that we established beforehand.

It still is the case that, for example, this protocol in a shared secret system is published.

And so people can go and try and analyze the security of this thing.

The community can look at what the math is that is going on.

Let's look at a simple example of a shared key system.

This is an approach called a one-time pad. One-time pad is a very simple example of an encryption protocol that is essentially cryptographically unbreakable.

That is it may be possible to break it through other means but it is sort of provably not possible to break this through some sort of mathematical analysis attack. One-time pad depends on the ability of a source of truly random bits.

I need some way to generate a set of bits which are sort of completely random.

And doing that is tricky, but let's suppose that we can do it.

What one-time pad says, we're going to call this sequence of bits k, that's going to be our key, and this key and the one-time pad approach is not going to be short.

This key is going to be very, very long.

It's going to be as long as all of the messages that we possibly want to send over all of time.

Suppose that Alice and Bob, the way they generate this is Alice writes a set of random bits onto a CD, writes 650 megabytes of random bytes onto a CD and gives that to Bob and they agree that this is the key that they are going to use over time.

We have m and k coming in.

They are being combined together.

They are being transmitted over the Internet.

And then, on the other side, they are being decoded also with k where these two ks are equal in this case and we've got m coming out.

This plus operation here is an XOR.

This plus with a circle is XOR.

If you were to remember what XOR does, the definition of XOR is given two bytes, zero, one, two things that were XORing together, two bytes that are either zero or one, the XOR of two zeros is zero, the XOR of zero and one is one and the XOR of two ones is zero.

XOR has this nice property.

This XOR is a byte-wise operation.

This says the first byte in k is XORed with the first bid in m and the second byte in k is XORed with the second byte in m and so on.

And the same thing happens over here.

XOR has the nice property that m XOR k XOR k is equal to m.

You can verify that that's true if you look at a simple example.

What happens is when the stream m comes in and gets XORed with k, the encrypted message that is traveling over here essentially looks like a random byte string because k is a random byte string.

And this m, no matter what m is, when it is XORed with a random byte string you're going to get something that looks like a random byte string coming out.

And then on this other side, though, we also have access to the same byte string that was used, and now we can decrypt the message.

So only if somebody knows exactly what k is can they decrypt the message.

This approach is hard to make work in practice because it requires the availability of a large amount of random data.

One thing that you can do is use a random number generator that is seeded with some number and then use that random number generator to begin to generate the sequence of bytes.

Of course, that has the problem now which is somebody can discover the seed or somebody can guess which seed you used because the random number generator used is not very good.

They may be able to break your protocol.

But on-time pad is a nice example of something which is sort of a cryptographic protocol that is known to work pretty well.

What I want to do, just with the last five minutes, is talk about one specific cryptographic protocol which is called the RSA protocol.

And we probably won't have time to go through the details of how RSA actually works, but RSA is a public key protocol.

And let me just quickly show you what RSA uses and then we will skip over DES here.

What RSA does is says we are going to take two numbers, p and q which are prime.

This protocol is in the book.

It is in appendix one of the chapter so you don't need to copy it down word for word if you don't want to.

If we have p and 1 are primes, we pick two numbers p and q which are prime, and then we generate some number n which is equal to p times q and another number z which is equal to p minus one times q minus one.

And then pick another number e which is relatively prime to z.

So that relatively prime just means that e doesn't divide z evenly.

So z is not divisible by e.

And we pick a number d such that e times d is equal to one, as long as the product [is modulo z?].

If you take e times d and you take the modulus with z the value should be one.

If you pick a set of numbers that satisfy this property then we can define the public key to be e, n and the private key to be d, n.

And these numbers have this magic property.

And then we are going to be able to, with this, encrypt any message that is up to n in length.

So, in general, we are going to want n and p and q to be some set of very large numbers.

They are going to be hundreds of bytes long.

We are going to be able to encrypt any message that is up to n bytes long, it is up to size of n.

So we may have to break the messages up into chunks that are size n or smaller.

This has this magic property now that if we encrypt the data in the following way, to encrypt a message we take m to the power of e and then take the whole thing module n.

Now, transmit that message c across the network.

Now, to decrypt, we take that encrypted thing and take it to the power d and then take the whole thing modulo n, we get the original message out at the end.

The mathematics of understanding why this work turned out to be fairly subtle and sophisticated.

There is a brief outline of it in the paper, but if you really want to understand this it's the kind of thing that requires an additional course in cryptography.

We're not going to go into the details of the mathematics, but the idea is suppose we pick p to be 47 and q to be 49, two prime numbers, generate n to be 2773, just the product of those, z then is 2668.

And now we pick two numbers e and d.

We pick these numbers just by using some searching for these numbers somehow over the set of all possible numbers we could have picked.

So we have these two numbers e and d which satisfy this property.

That is 2668 is not divisible by 17, and 17 times 157 modulo z, modulo 2668 is equal to one.

And you have to trust me that it true.

Now, suppose we have our message is equal to 31. If we compute now C is equal to this thing, 31 to the 17th modulo 2773, we get 587.

And then, sort of magically at the end, we reverse this thing and out pops our message. What we see here is a public key protocol.

What we say here is the public key is equal to this combination of e and n and the private key is equal to this combination of d and n.

Suppose that only Alice knows the private key and that Bob knows the public key and everybody else in the world, for example, knows what the public key is.

Now, what we can do is Alice can encrypt the message with her private key which nobody else knows.

And then using the public key everybody else can go ahead and decrypt that message.

And by decrypting that message they can be assured that the only person that could have actually created this message to begin with is somebody who had access to Alice's private key.

So they can authenticate that this message came from Alice.

This protocol also has a nice side effect which is that it is reversible.

If Bob encrypts a message using Alice's public key, Alice can decrypt that message, and only Alice can decrypt that message using her private key.

We will talk more about these properties next time.

And take care.