OK, so today we're going to continue our discussion of protection.

Remember, we have our three protection primitives that we've been talking about, authentication, authorization and confidentiality.

Today we're mostly going to focus on authorization and confidentiality.

Remember, we have seen, and we are going to continue to rely on this set of cryptographic primitives.

So the cryptographic primitives that we talked about are sign and verify.

And we said that sign accepts a message and some key.

And verify accepts the message and some k.

Sign accepts a message and a key and outputs some signature.

Verify accepts the message and the signature and some other key k2 and tells us whether or not it believes that the message that this m actually corresponds to this signature.

And we talked about how if k1 is equal to k2 this is a shared key system.

And if k1 is not equal to k2 we typically call this a public key system.

We also talked about our encrypt and decrypt primitives.

And we said that encrypt accepts some message m and some key k1 and outputs some encoded message C.

And decrypt takes C and some k2 and gives back m.

Again, we have the same separation between shared key and public key, private key.

So, if you remember last time, we were talking about this protocol that was a protocol for establishing a secure communication channel.

We got as far as talking about how we would actually authenticate a user.

We talked about the process for authentication.

And then we were using our authentication process to build up a protocol for setting up a secure communication channel.

The way this is going to work is that we are going to use a public key cryptography.

We are going to use our encryption with a public key to exchange a shared key.

And then we are going to use the shared key to encrypt our communication.

We went through this example with the Denning-Sacco Protocol which I will quickly review.

And, remember, I told you that this protocol is broken.

And we got as far as not quite figuring out how it was broken.

So let's see if we can figure out how it was broken.

Again, we said the idea is Alice first sends a message to Charles.

And, remember in our model, Charles is this certificate authority, the guy who both Alice and Bob know and trust.

So Alice sends a message to Charles saying hey, I would like to get these certificates for Alice and Bob so that I can establish this clear communication channel.

Charles goes ahead and agrees with this and sends a message back which consists of these two certificates, one certificate for Alice and one certificate for Bob.

Both of these certificates have been signed with Charles' private key.

And because they are signed with Charles' private key that presumably means that only Charles could have possibly generated these certificates.

And there are two certificates, and each certificate contains the name of the principle that the certificate belongs to, the public key of that principle and then some timestamp which, remember, we said we want to use to make sure that these certificates are fresh.

And now what happens is that Alice, in order to establish this communication with Bob, needs to exchange or needs to propose a shared key that she can go ahead and use with Bob.

What she does is she sends her certificate, which is something that Bob should be able to use to validate that she is, in fact, who she said she is.

And, remember, she cannot forget this because it's signed with Charles' private key.

And she also goes ahead and sends this proposed key.

And the way she encodes the proposed key is to sign it with her private key so that Bob knows that it actually came from her.

And then she encrypts it with Bob's public key so that only Bob can actually go ahead and decode this message.

Let's think for a minute.

We started to get towards figuring out what's wrong with this protocol.

And the way that we started to talk about it was, well, what properties would we like a protocol like this to have?

We said one property we would like is for it to have freshness.

What freshness means is that the protocol shouldn't be susceptible to replay attacks.

That is somebody shouldn't be able to overhear a message and then replay it sometime much later in order to sort of cause that message to happen again.

So, even if that person cannot actually look inside the contents of the message, they may be able to get something to happen simply by replaying the message.

So freshness is something we want our protocols to have.

And, in this case, we are achieving freshness hopefully by including this T which is a timestamp in the message.

As I said last time, getting the timestamp to actually work correctly is something that you have to be careful about.

And this is described in Section E of the notes.

The first property is freshness.

The second one is appropriateness.

That just means that this message is, in fact, for the particular sender, it applies for the particular sort of context in which the message is being used.

So the people who are receiving the message, for example, are actually the intended recipients of the message.

And then the third property we said we want is forward secrecy.

And this just means that it will be possible to switch to a new set of keys at some later time if, for example, the keys get breached so we have a way of changing the protocol.

This protocol that we have also achieves forward secrecy.

The problem it has is with appropriateness.

Let me show you what I mean by this protocol being inappropriate.

Once we've exchanged this kAB now we can go ahead and Bob and Alice can presumably communicate with each other by encrypting messages to one another with this kAB key that they've exchanged.

Let's see what the problem with this is.

Remember, Alice sends this message three to Bob.

Now here is what the issue is.

The issue has to do with this piece of information that Alice has encrypted and signed.

And the problem is that this message does not satisfy the appropriateness constraint.

In particular, it doesn't have any context about which conversation this key is meant to apply to.

So let me show you a way in which Bob might exploit this.

Suppose Bob sends a message to Charles.

And what he sends in this message is I am Alice and I would like to set up a conversation with you and here is my public key and here is a bit of information that I've signed using my private key that is a key I would like to use for our conversation.

So Charles sees this thing and says oh, yes, Alice signed this with her private key.

He can check with the certificate authority and see that, in fact, this message appears to be authentic.

And now he may go ahead and send a message back to Bob that is signed with this kAB.

So Charles has been fooled into thinking that Bob is, in fact, Alice and goes ahead and starts to establish some communication with him.

Again, the issue is because this encrypted and signed thing doesn't have any information about what conversation this key is meant to apply to.

Alice hasn't said that this is only a key that applies to a conversation between Alice and Bob.

Now, anybody who overhears this can use this key to establish a conversation that appears to be originating with Alice that, in fact, is not.

Let me show you how we go ahead and fix this.

This is just exactly the protocol that I showed you before.

And the way that we're going to fix this is really pretty much as I've just described.

We are going to take this big nasty statement that we had here, and we're going to basically sign and encrypt the whole message instead of simply signing and encrypting the one little piece of this message which was the proposed key.

We are going to sign and encrypt the whole thing together, including Alice's public key and Alice's certificate.

And we are also going to include a sender and receiver in this entire message.

Now we said we add A and B as two members of this entire message.

And what this allows us to do is to guaranty that because Alice has signed this entire thing, Bob isn't able to generate a new version of this thing, or Bob isn't able to use this thing to spoof a conversation with Charles.

Bob can go ahead and try and send this message for Charles, but Charles could look at this thing and obviously tell that this key was not for a conversation between himself and Alice but instead was for a conversation between Bob and Alice.

And there is no way that Bob could spoof the generation of this whole thing because he doesn't have Alice's private key so he couldn't sign this message.

That wraps up most of our discussion, the extent of the discussion from last time.

What we saw last time basically is a way to do authentication, and then at the very end we talk about how we can use authentication to establish the secure communications channel.

Establishing a secure communications channel relies on sort of the confidentiality piece of this story, and I just want to spend a few minutes talking about what confidentiality is and how it works.

Confidentiality, obviously, is the protection of information exchange between two people or two principles, say Alice and Bob.

And the idea is to make it so that somebody who is overhearing this communication wouldn't actually be able to tell what the contents of the message that was overheard were.

The idea is suppose Alice generates some message m, puts it into an encryption box, encrypts it with some key k1 and generates a new message C which gets sent over the Internet to Bob.

On the other side, Bob feeds this thing into a decryption box, decrypts it with k2 and gets m out.

And this has the properties that simply knowing C makes it very hard to drive m.

Given C, driving m is very hard to do.

That is because of the difficulty of sort of breaking these cryptographic protocols that we talked about the first couple times, but given k2 and C it is possible to drive m.

The only way that you can do this is by applying some sort of attack on the cryptographic mechanism hopefully.

But this property, given k2 you can derive C, it is relatively easy for the receiver to go ahead and figure out what the message is.

And since we are using these encrypt and decrypt boxes, obviously, k1 could be equal to k2, in which case we would be using a shared key approach.

And k1 and k2 can be different, in which case we are using public key like we talked about.

What we did in the secure communication channel protocol that we just talked about was to both authenticate and establish confidentiality.

We had both confidentiality and authentication, so at the end of this protocol Alice and Bob have authenticated with each other and they have confidentiality because they've exchanged this key kAB that they can use to have a private conversation with one another.

This is going to be a common thing that we are going to want to be able to do, confidentiality plus authentication.

And we see the way that we do that is exactly the pattern that we have going on here in this example.

You take a message and you sign it and then you encrypt it.

So you have your message m.

It is signed with some key which we will call kconf for confidentiality.

Sorry.

It is encrypted with kconf.

And then you're going to sign this whole thing with some key kauth which is your key for authorization.

And it is OK, in this case, for these two things, this sign and encrypt can be done in either order.

I can sign the message and then encrypt it or encrypt it and then sign it.

It works out just fine.

This pattern is going to be a common one.

Often times, when we're building up a secure system, what you want to do is first authenticate.

You want to both guaranty that the person you are talking to, you want to know who the person you are talking to is, and you also want the communication with that person to be secure.

We talked at the beginning, when we first starting talking about authentication we talked about a simpler example where you might just want to authenticate.

I had the example, suppose you're purchasing something and you don't care if somebody knows that you purchase it, like giving money to Save the Whales.

That example, you're giving money to save the whales.

All you care about is that Save the Whales can actually authenticate your message, that the message is well-formed from the point of view that Save the Whales doesn't believe that you're giving them $10,000 instead of a $100 and that Save the Whales knows who you are so get credit for giving them that money, but you don't actually maybe care whether that message is encrypted.

Another common thing you are going to do is just sign a message with kauth.

It is less common to see, well, the other thing you might wonder is how often do people just want confidentiality?

Just establishing confidentiality is sort of less common.

You would see that less frequently.

Typically, people want to either do confidentiality and authentication or just authentication.

But just confidentiality is a little bit of a weird case because having a private conversation with somebody who you don't know there aren't that many cases where you want it.

You could imagine an anonymous communication system, for example, where you don't actually want the other person to know anything about who you are.

But that's a little bit of an unusual situation.

These are kind of the two major forms of private communication that we have talked about so far.

And we talked about a few little other details.

This example illustrated a few other little details that we need to make sure we take care of.

One of them was this freshness constraint.

In this example, we said this is e.g., the addition of T to this example up here.

So, when we add the timestamps, that makes it difficult for somebody to apply a replay attack against us.

The other thing that we might want to do is add T to m.

This is freshness.

This is going to add timestamp to m.

Now, the other thing we might want is some way to guaranty appropriateness.

And, what we're going to do for appropriateness, what we need to do to make sure the message is appropriate is we need to add context to the message so we need to add some information that specifies who is originally involved in the message.

These are sort of the two major kinds of protection that we have, and we need to make sure that our protocols we use sort of provide these kinds of guarantees.

Now what I want to do is talk a little bit about authorization.

We've seen authentication and a bit of confidentiality and we've talked about authorization, but I want to talk about it in the context of an example.

Because you guys know something about authorization techniques.

You guys have all seen, for example, passwords for logging into a system.

There is no big surprise about how passwords work.

There is some list of passwords.

And, when a user tries to log in, the system checks to see if the person is in the list.

They typically take the hash of the password that is typed in and check to see if the hash of the password is in the password list.

This is described in detail in the text and we will go over it a little bit today, but what I want to do is sort of talk about how we fit all three of these things, authentication, authorization and confidentiality together into one system as we talk about authentication.

We are going to use an example.

Which is the Web.

Suppose that we were in a situation where we have some browser B communicating with some Web server W over a secure communication channel.

And, if you like, you can think of the secure communication channel as having been established by a protocol like the one I've shown here.

In practice on the Web, there is a common protocol that is used to establish the secure communication channel called SSL, secure sockets layer.

This channel has been authenticated and is confidential.

It is typically the case, for example, that there may be some certificate authority, for example, which B has used to discover keys for W, to discover a private key to use to talk to the Web server, W.

There is a question, though, that we haven't really got at which is how does W know that B is authorized to access?

And we hinted at this on the first lecture about security, but this is the question that we want to try and address today in a little more detail.

The issue is that once this protocol is established, these two guys have exchanged a key with each other, this kAB or kBW.

And this is a shared key that these guys can communicate with each other.

But basically all that W knows about B is that this is somebody who has this key.

This is the same B who initiated this connection with me.

W may not have actually checked to go ahead and see what it is on the server, what services on WB has access to.

So W needs to go ahead and figure out what it is that B can access.

And these two sort of check all the accesses that B tries to make on W to make sure that it is authorized to do so.

If you remember back to the very first lecture, we talked about how authentication and authorization work.

We sort of said there are two steps.

Or, in this case, we are going to actually talk about three steps.

We have three steps or let's call it three authorization functions.

And these functions actually sort of share something in common with authentication because authenticating and authorizing are sort of intertwined together.

You will see what I mean.

The first step is we need some rendezvous.

When we talk about this, this is the way that two principles initially set up the access rights to the particular server.

For example, this is you going to your system administrator and telling your system administrator that you would like an account.

He creates an account, he creates a home directly for you and then you have the rights to access any of the files that are in your home directly.

So think of rendezvous as setup.

Account creation, you can log onto Amazon dot com and create an account, for example.

Then there is some other step which is this verification step.

And verification is simply making sure that when you reconnect to the system you are allowed to connect to the things that you want to connect to.

Typically, in an authorization system, we talk about this thing mediating your communication with the Web server.

If I log onto Amazon dot com and say that I would like to log in as a particular user, this verification step is going to make sure that I have the appropriate credentials to log on as that user.

The final thing I might want to do is to revoke.

I might want to simply remove a user from the system or make it so that the user is no longer a part of the system.

There are two widely sort of used approaches for authentication that sort of do different things at these different steps.

The two approaches are called lists and tickets.

Authorization.

We have these two approaches.

The first one we are going to call lists and the second one is called tickets.

And we have these three steps.

We have our setup step, we have our mediation step and we have our revocation step.

Lists are something you may be familiar with.

One way that we might authorize whether or not a user is allowed to access a system is to check some list of all the users who are allowed access, and we may check that user's credentials.

For example, if you go to a party and that party is invite only and requires you to show your MIT ID at the door, you show up at the door, you show them your MIT ID and then that lets you in.

And now you've been sort of authorized to access the party.

The other approach is a ticket-based approach.

This is an approach where instead of having a list of people and checking credentials whenever the person wants access, instead you have some ticket which lets anybody who has that ticket have access.

This is a party is invitation only.

You get an invitation in the mail.

And now anybody who has that invitation can go ahead and go to the party.

You guys are all familiar with systems that work like that, baseball games, carnival games or whatever it is.

You get tickets and you use those tickets to get access to something.

And anybody who has those tickets can use them.

There is no checking of your sort of credentials every time you try to use them.

As we're going to see, and we will talk about how this works on the Web, it is often the case that you use one of these list-based authentication systems to decide who we should issue tickets to.

And then you give a bunch of people tickets and anybody can do whatever they want with those tickets.

Let's talk about sort of the properties of these things and a little bit about how they work in the context of a computer system.

The setup process and list is obviously you add someone to a list.

Passwords typically are done with list-based systems.

The idea is when you add somebody to a password list you input their name or their account name and a cryptographic hash of the password that they typed in.

Usually, you want them to type the password in, in some secure fashion.

You put the cryptographic hash of that password there.

And then, in order to mediate, in order to actually verify, for example, that the password is going to be correct, you are going to search the list and you are going to verify that the password that they present, in fact, hashes to the stored hash in the password file.

You are going to search the list and you're going to also do this sort of check credentials step and make sure the password is what it was supposed to be.

It is relatively easy then, in this environment, to revoke somebody's access.

You can remove them from list.

And then, in that case, well, they won't be able to use the system anymore.

Tickets.

The idea in a ticket-based system is that you are going to generate a ticket.

And usually what that means in a computer system, or the simplest way to generate a ticket in a computer system is to make a sort of hard to guess number.

For example, this might be a hash of something or it might just be a big random number.

You want it to be something that if somebody picks another random number out of thin air they don't have a very high probability of guessing a number that actually allows them to access the system, but anybody who has this ticket should be able to access the system relatively easily.

Because these tickets are just, for example, a big random number, users typically should feel free to share these, might be able to exchange these tickets with each other.

They should be able to share them with other people and use them for access.

Whereas, users typically are not going to share their passwords with other people.

Tickets are a way that a user can handoff sort of the authority, the right to access a system to some other user, they can delegate.

Now, to mediate we are just going to look up, this is just going to be a table lookup.

We just need to make sure that this is a valid number that was, in fact, given to us.

And, typically, we don't have to go and check credentials.

The user doesn't have to supply a password when they supply us with a ticket.

And then, finally, in order to revoke, well, we might be able to invalidate a ticket.

You might be able to say, for example, remove her from the table.

That works fine if we're OK with getting rid of an entire ticket.

But suppose, for example, that I want to revoke the right of a single user to access the system.

That is very hard to do in a ticket-based system.

Suppose that you guys decide that you no longer want me looking at some discussion board about 6.033 that you've set up.

And suppose because you're writing bad things about me and you say, well, we don't want the professor to be able to access this anymore.

So you try and revoke my access.

If you've been using a ticket-based system that is going to be very hard unless you invalidate that ticket.

And there are many people who are sharing the same ticket.

That's going to be hard to deny me access because you're going to have to revoke that ticket and then go to everybody else who might be sharing that ticket and issue them a new one.

Whereas, in this sort of list-based system you could have just removed my account.

This has an analogy in the real world.

For example, door locks are sort of like a ticket-based system.

You generate a physical key for something.

You could give that out to many people, and then many people can have access to your house.

The problem is if you want to revoke access to a single person then you have to sort of go and either collect the keys from everybody, which may not be feasible, or change the door locks which denies everybody access until you reissue keys to the people who need them.

If you like, you can think of a ticket-based system sort of like being door locks in a house.

Oftentimes lists are more formally called ACLs or access control lists.

You will see this term used in the text, in fact, as well.

And tickets are sometimes called capabilities.

Capabilities are sort of the right to access some resource on the computer that can be delegated from one user to the other or one principle to the other.

And, in practice, what we see both, as I said, in the real world and in computer systems is that we are often going to use this list-based mechanisms to decide who to hand tickets out to.

In the case of, OK, who are we going to invite to our party, who are we going to send invitations to in the mail, well, you might have some list that you go ahead and assemble based on either people who you know to be MIT students or people who meet some particular set of eligibility criteria.

That is the sort of process of checking a list.

And then, once you've assembled the list, you are going to generate these tickets and send them out, and then anybody can use those tickets.

This same analogy sort of holds true in the Internet, so let's look at how we might build up an authorization system on top of our secure channel between our browser and our Web server.

We have our B talking over our secure channel of our Web server.

And the idea is going to be as follows.

This Web server has some service that the user wants to try and access, and we are going to have this guard.

Remember, we talked about this guard model before.

There is going to be some guard that sits in front of this service that is in charge of authorizing B to access this service.

And then there is some authorization module that runs on B.

The idea is that B is going to initiate a connection and then guard is going to send a message to the authorization module on B saying who are you, what are your credentials to access this system?

This is going to be a form of list-based access.

Now, this module on B is going to go ahead and send a message back that says my name is, whatever my name is, as well as some key, some credential like, for example, a password that is their password for accessing this system.

Notice that these two guys do not need to worry about sort of protecting this channel, as far as the rest of the Internet is already concerned, because they have already established this secure channel upon which they can do this exchange of information.

So they are hopefully not worried about other people overhearing this message.

They may not need to protect this information that is being transmitted in the same way because they have already got the secure communication channel that they have established at the lower levels of this thing.

All of this communication is going over this secure communication channel between these two guys.

This is a layer that is built on top of this secure communication layer.

Now what happens is that what the guard is going to do is look up this name and password, for example, in some access control list and use that to determine whether or not this person has the rights to access and what they have the rights to access to.

And he is going to generate for this person a set of tickets that represent the sort of services that this person can access on this system.

He is going to have some table of tickets, and this table is going to have the ticket number and the resource, for example, that the user is allowed to access.

So he is going to send back some ticket number authNo back to the user.

So we are going to add authNo.

And then maybe this gives him the right to access some account, B's account information.

Now, with this authorization information, this is the ticket.

Now every time that B wants to go ahead and access his account, for example, he can read account B and then he just passes the ticket.

And he can use this ticket over and over and over again to access the account for as long as the ticket is valid.

Often times there will be a timeout associated with tickets.

But as long as the ticket is valid he can go ahead and supply this ticket in order to be able to access the resource that the ticket gives him access to.

And he doesn't have to sort of re-authenticate himself.

He doesn't have to represent his credentials every time he wants to access this resource on his account on the Web server.

Are the HKN people here?

Yes.

OK.

We need to do HKN.

Why don't you just give me one more minute to wrap up and then we can go ahead and do the HKN review.

This protocol, I mentioned that this is sort of like SSL, or the SSL works in sort of this way.

On the Internet you may have heard about cookies.

Cookies on the Web are essentially a kind of ticket.

The idea with most cookie-based systems is you log in with a password.

And then, when you log in with that password, the system issues you a cookie.

And then when you want to re-access the system you simply supply the cookie.

If you were to look at the contents of a prototypical cookie, it would typically have something like, for example, the user's name and some timeout, a valid period for the cookie, as well as a hash of the user's name, that timeout and some, for example, random number which is only known over here at the service.

The service is going to protect this random number and is going to hash it together with the user's ID and this timeout.

And so now, when this cookie is supplied, it is going to be hard for users to generate fake cookies, but these cookies are going to allow the user to go ahead and access the services on W without having to re-authenticate every time.

And you can, in fact, in some cookie systems share those cookies.

You could copy the cookie from one browser to another, and you might be able to reuse that cookie to access the system for as long as the cookie is valid.

Cookies typically timeout after an hour or two, and so then you would have to re-authenticate with the system.

That is all I wanted to really say about authentication.

What we are going to do is talk about some sort of advanced protocols next time.

I want to remind you guys that there is no class on this Wednesday so don't come here.

The next class is next Monday.

And that is it.

Good luck finishing your DP2 and have fun doing HKN reviews.