MITOCW | ocw-6.033-lec23

0:00:00 OK, so today we're going to continue our discussion of 0:00:06.691 protection. Remember, we have our three 0:00:11.489 protection primitives that we've been talking about, 0:00:17.929 authentication, authorization and 0:00:21.969 confidentiality. 0:00:25 0:00:30 Today we're mostly going to focus on authorization and 0:00:35.282 confidentiality. Remember, we have seen, 0:00:39.169 and we are going to continue to rely on this set of 0:00:44.152 cryptographic primitives. So the cryptographic primitives 0:00:49.734 that we talked about are sign and verify. 0:00:53.72 And we said that sign accepts a message and some key. 0:01:00 And verify accepts the message and some k. 0:01:04.702 Sign accepts a message and a key and outputs some signature. 0:01:11.469 Verify accepts the message and the signature and some other key 0:01:18.58 k2 and tells us whether or not it believes that the message 0:01:25.232 that this m actually corresponds to this signature. 0:01:32 And we talked about how if k1 is equal to k2 this is a shared 0:01:37.544 key system. And if k1 is not equal to k2 we 0:01:41.425 typically call this a public key system. 0:01:45.029 We also talked about our encrypt and decrypt primitives. 0:01:50.112 And we said that encrypt accepts some message m and some 0:01:55.194 key k1 and outputs some encoded message C. 0:02:00 And decrypt takes C and some k2 and gives back m. 0:02:03.692 Again, we have the same separation between shared key 0:02:07.692 and public key, private key. 0:02:09.769 So, if you remember last time, we were talking about this 0:02:14.076 protocol that was a protocol for establishing a secure 0:02:18.153 communication channel. We got as far as talking about 0:02:22.153 how we would actually authenticate a user. 0:02:25.307 We talked about the process for authentication. 0:02:30 And then we were using our authentication process to build 0:02:33.831 up a protocol for setting up a secure communication channel. 0:02:38 0:02:48 The way this is going to work is that we are going to use a 0:02:55.828 public key cryptography. We are going to use our 0:03:02.171 encryption with a public key to exchange a shared key. 0:03:10 And then we are going to use the shared key to encrypt our 0:03:15.053 communication. We went through this example 0:03:18.776 with the Denning-Sacco Protocol which I will quickly review. 0:03:24.007 And, remember, I told you that this protocol 0:03:27.819 is broken. And we got as far as not quite 0:03:31.365 figuring out how it was broken. So let's see if we can figure 0:03:36.977 out how it was broken. Again, we said the idea is 0:03:40.25 Alice first sends a message to Charles. 0:03:42.84 And, remember in our model, Charles is this certificate 0:03:46.522 authority, the guy who both Alice and Bob know and trust. 0:03:50.34 So Alice sends a message to Charles saying hey, 0:03:53.477 I would like to get these certificates for Alice and Bob 0:03:57.227 so that I can establish this clear communication channel. 0:04:02 Charles goes ahead and agrees with this and sends a message 0:04:06.319 back which consists of these two certificates, 0:04:09.67 one certificate for Alice and one certificate for Bob. 0:04:13.617 Both of these certificates have been signed with Charles' 0:04:17.787 private key. And because they are signed 0:04:20.691 with Charles' private key that presumably means that only 0:04:24.861 Charles could have possibly generated these certificates. 0:04:30 And there are two certificates, and each certificate contains 0:04:33.927 the name of the principle that the certificate belongs to, 0:04:37.659 the public key of that principle and then some 0:04:40.604 timestamp which, remember, we said we want to 0:04:43.485 use to make sure that these certificates are fresh. 0:04:46.758 And now what happens is that Alice, in order to establish 0:04:50.424 this communication with Bob, needs to exchange or needs to 0:04:54.155 propose a shared key that she can go ahead and use with Bob. 0:04:59 What she does is she sends her certificate, which is something 0:05:03.589 that Bob should be able to use to validate that she is, 0:05:07.652 in fact, who she said she is. And, remember, 0:05:10.888 she cannot forget this because it's signed with Charles' 0:05:15.026 private key. And she also goes ahead and 0:05:17.961 sends this proposed key. And the way she encodes the 0:05:21.798 proposed key is to sign it with her private key so that Bob 0:05:26.162 knows that it actually came from her. 0:05:30 And then she encrypts it with Bob's public key so that only 0:05:34.484 Bob can actually go ahead and decode this message. 0:05:38.273 Let's think for a minute. We started to get towards 0:05:42.139 figuring out what's wrong with this protocol. 0:05:45.541 And the way that we started to talk about it was, 0:05:49.252 well, what properties would we like a protocol like this to 0:05:53.737 have? We said one property we would 0:05:56.365 like is for it to have freshness. 0:06:00 What freshness means is that the protocol shouldn't be 0:06:02.814 susceptible to replay attacks. That is somebody shouldn't be 0:06:05.946 able to overhear a message and then replay it sometime much 0:06:09.026 later in order to sort of cause that message to happen again. 0:06:12.212 So, even if that person cannot actually look inside the 0:06:15.079 contents of the message, they may be able to get 0:06:17.575 something to happen simply by replaying the message. 0:06:20.283 So freshness is something we want our protocols to have. 0:06:23.203 And, in this case, we are achieving freshness 0:06:25.539 hopefully by including this T which is a timestamp in the 0:06:28.513 message. As I said last time, 0:06:31.538 getting the timestamp to actually work correctly is 0:06:35.384 something that you have to be careful about. 0:06:38.692 And this is described in Section E of the notes. 0:06:42.307 The first property is freshness. 0:06:44.692 The second one is appropriateness. 0:06:47.23 That just means that this message is, in fact, 0:06:50.692 for the particular sender, it applies for the particular 0:06:54.923 sort of context in which the message is being used. 0:07:00 So the people who are receiving the message, for example, 0:07:04.107 are actually the intended recipients of the message. 0:07:07.848 And then the third property we said we want is forward secrecy. 0:07:12.396 And this just means that it will be possible to switch to a 0:07:16.65 new set of keys at some later time if, for example, 0:07:20.317 the keys get breached so we have a way of changing the 0:07:24.205 protocol. This protocol that we have also 0:07:27.139 achieves forward secrecy. The problem it has is with 0:07:31.852 appropriateness. Let me show you what I mean by 0:07:35.13 this protocol being inappropriate. 0:07:37.482 Once we've exchanged this kAB now we can go ahead and Bob and 0:07:41.757 Alice can presumably communicate with each other by encrypting 0:07:46.104 messages to one another with this kAB key that they've 0:07:49.881 exchanged. Let's see what the problem with 0:07:52.802 this is. Remember, Alice sends this 0:07:55.225 message three to Bob. Now here is what the issue is. 0:08:00 The issue has to do with this piece of information that Alice 0:08:04.054 has encrypted and signed. And

the problem is that this 0:08:07.635 message does not satisfy the appropriateness constraint. 0:08:11.351 In particular, it doesn't have any context 0:08:14.121 about which conversation this key is meant to apply to. 0:08:17.77 So let me show you a way in which Bob might exploit this. 0:08:21.554 Suppose Bob sends a message to Charles. 0:08:25 And what he sends in this message is I am Alice and I 0:08:29.126 would like to set up a conversation with you and here 0:08:33.253 is my public key and here is a bit of information that I've 0:08:37.857 signed using my private key that is a key I would like to use for 0:08:42.936 our conversation. So Charles sees this thing and 0:08:46.666 says oh, yes, Alice signed this with her 0:08:49.761 private key. He can check with the 0:08:52.38 certificate authority and see that, in fact, 0:08:55.793 this message appears to be authentic. 0:09:00 And now he may go ahead and send a message back to Bob that 0:09:03.774 is signed with this kAB. So Charles has been fooled into 0:09:07.353 thinking that Bob is, in fact, Alice and goes ahead 0:09:10.607 and starts to establish some communication with him. 0:09:13.926 Again, the issue is because this encrypted and signed thing 0:09:17.7 doesn't have any information about what conversation this key 0:09:21.605 is meant to apply to. Alice hasn't said that this is 0:09:24.924 only a key that applies to a conversation between Alice and 0:09:28.698 Bob. Now, anybody who overhears this 0:09:32.073 can use this key to establish a conversation that appears to be 0:09:36.221 originating with Alice that, in fact, is not. 0:09:39.165 Let me show you how we go ahead and fix this. 0:09:42.109 This is just exactly the protocol that I showed you 0:09:45.454 before. And the way that we're going to 0:09:47.996 fix this is really pretty much as I've just described. 0:09:51.542 We are going to take this big nasty statement that we had 0:09:55.288 here, and we're going to basically sign and encrypt the 0:09:58.901 whole message instead of simply signing and encrypting the one 0:10:02.982 little piece of this message which was the proposed key. 0:10:08 We are going to sign and encrypt the whole thing 0:10:11.126 together, including Alice's public key and Alice's 0:10:14.386 certificate. And we are also going to 0:10:16.781 include a sender and receiver in this entire message. 0:10:20.241 Now we said we add A and B as two members of this entire 0:10:23.9 message. And what this allows us to do 0:10:26.361 is to guaranty that because Alice has signed this entire 0:10:30.02 thing, Bob isn't able to generate a new version of this 0:10:33.613 thing, or Bob isn't able to use this thing to spoof a 0:10:37.072 conversation with Charles. Bob can go ahead and try and 0:10:42.1 send this message for Charles, but Charles could look at this 0:10:46.6 thing and obviously tell that this key was not for a 0:10:50.424 conversation between himself and Alice but instead was for a 0:10:54.85 conversation between Bob and Alice. 0:10:58 And there is no way that Bob could spoof the generation of 0:11:02.11 this whole thing because he doesn't have Alice's private key 0:11:06.365 so he couldn't sign this message. 0:11:08.673 That wraps up most of our discussion, the extent of the 0:11:12.567 discussion from last time. What we saw last time basically 0:11:16.677 is a way to do authentication, and then at the very end we 0:11:20.788 talk about how we can use authentication to establish the 0:11:24.826 secure communications channel. Establishing a secure 0:11:29.272 communications channel relies on sort of the confidentiality 0:11:32.848 piece of this story, and I just want to spend a few 0:11:35.878 minutes talking about what confidentiality is and how it 0:11:39.212 works. 0:11:40 0:11:45 Confidentiality, obviously, is the protection of 0:11:47.35 information exchange between two people or two principles, 0:11:50.2 say Alice and Bob. And the idea is to make it so 0:11:52.549 that somebody who is overhearing this communication wouldn't 0:11:55.5 actually be able to tell what the contents of the message that 0:11:58.549 was overheard were. The idea is suppose Alice 0:12:02.403 generates some message m, puts it into an encryption box, 0:12:07.788 encrypts it with some key k1 and generates a new message C 0:12:13.269 which gets sent over the Internet to Bob. 0:12:17.115 On the other side, Bob feeds this thing into a 0:12:21.442 decryption box, decrypts it with k2 and gets m 0:12:25.769 out. And this has the properties 0:12:28.75 that simply knowing C makes it very hard to drive m. 0:12:35 Given C, driving m is very hard to do. 0:12:37.854 That is because of the difficulty of sort of breaking 0:12:41.867 these cryptographic protocols that we talked about the first 0:12:46.419 couple times, but given k2 and C it is 0:12:49.274 possible to drive m. The only way that you can do 0:12:52.978 this is by applying some sort of attack on the cryptographic 0:12:57.53 mechanism hopefully. But this property, 0:13:01.455 given k2 you can derive C, it is relatively easy for the 0:13:05.902 receiver to go ahead and figure out what the message is. 0:13:10.35 And since we are using these encrypt and decrypt boxes, 0:13:14.716 obviously, k1 could be equal to k2, in which case we would be 0:13:19.568 using a shared key approach. And k1 and k2 can be different, 0:13:24.339 in which case we are using public key like we talked about. 0:13:30 What we did in the secure communication channel protocol 0:13:34.788 that we just talked about was to both authenticate and establish 0:13:40.273 confidentiality. We had both confidentiality and 0:13:44.365 authentication, so at the end of this protocol 0:13:48.283 Alice and Bob have authenticated with each other and they have 0:13:53.594 confidentiality because they've exchanged this key kAB that they 0:13:59.079 can use to have a private conversation with one another. 0:14:05 This is going to be a common thing that we are going to want 0:14:10.324 to be able to do, confidentiality plus 0:14:13.664 authentication. And we see the way that we do 0:14:17.635 that is exactly the pattern that we have going on here in this 0:14:23.14 example. You take a message and you sign 0:14:26.66 it and then you encrypt it. So you have your message m. 0:14:31.845 It is signed with some key which we will call kconf for 0:14:35.535 confidentiality. Sorry. 0:14:37.038 It is encrypted with kconf. And then you're going to sign 0:14:40.865 this whole thing with some key kauth which is your key for 0:14:44.76 authorization. And it is OK, 0:14:46.605 in this case, for these two things, 0:14:48.929 this sign and encrypt can be done in either order. 0:14:52.277 I can sign the message and then encrypt it or encrypt it and 0:14:56.309 then sign it. It works out just fine. 0:15:00 This pattern is going to be a common one. 0:15:02.175 Often times, when we're building up a secure 0:15:04.514 system, what you want to do is first authenticate. 0:15:07.179 You want to both guaranty that the person you are talking to, 0:15:10.443 you want to know who the person you are talking to is, 0:15:13.326 and you also want the communication with that person 0:15:16.1 to be secure. We talked at the beginning, 0:15:18.276 when we first starting talking about authentication we talked 0:15:21.539 about a simpler example where you might just want to 0:15:24.313 authenticate. I had the example, 0:15:26.893 suppose you're purchasing something and you don't care if 0:15:29.674 somebody knows that you purchase it, like giving money

purchasing something and you don't care if anybody knows that you purchase it, like giving money to Save 0:15:32.703 the Whales. That example, 0:15:33.895 you're giving money to save the whales. 0:15:35.782 All you care about is that Save the Whales can actually 0:15:38.464 authenticate your message, that the message is well-formed 0:15:41.294 from the point of view that Save the Whales doesn't believe that 0:15:44.422 you're giving them $10,000 instead of a $100 and that Save 0:15:47.253 the Whales knows who you are so get credit for giving them that 0:15:50.332 money, but you don't actually maybe care whether that message 0:15:53.311 is encrypted. Another common thing you are 0:15:56.765 going to do is just sign a message with kauth. 0:15:59.603 It is less common to see, well, the other thing you might 0:16:03.135 wonder is how often do people just want confidentiality? 0:16:06.603 Just establishing confidentiality is sort of less 0:16:09.63 common. You would see that less 0:16:11.522 frequently. Typically, people want to 0:16:13.792 either do confidentiality and authentication or just 0:16:17.009 authentication. But just confidentiality is a 0:16:19.783 little bit of a weird case because having a private 0:16:22.936 conversation with somebody who you don't know there aren't that 0:16:26.846 many cases where you want it. You could imagine an anonymous 0:16:32.196 communication system, for example, 0:16:34.613 where you don't actually want the other person to know 0:16:38.494 anything about who you are. But that's a little bit of an 0:16:42.594 unusual situation. These are kind of the two major 0:16:46.183 forms of private communication that we have talked about so 0:16:50.43 far. And we talked about a few 0:16:52.553 little other details. This example illustrated a few 0:16:56.288 other little details that we need to make sure we take care 0:17:00.535 of. One of them was this freshness 0:17:04.937 constraint. In this example, 0:17:07.58 we said this is e.g., the addition of T to this 0:17:12.083 example up here. So, when we add the timestamps, 0:17:16.685 that makes it difficult for somebody to apply a replay 0:17:21.874 attack against us. The other thing that we might 0:17:26.475 want to do is add T to m. This is freshness. 0:17:31.447 This is going to add timestamp to m. 0:17:34.262 Now, the other thing we might want is some way to guaranty 0:17:38.847 appropriateness. And, what we're going to do for 0:17:42.627 appropriateness, what we need to do to make sure 0:17:46.407 the message is appropriate is we need to add context to the 0:17:51.072 message so we need to add some information that specifies who 0:17:55.898 is originally involved in the message. 0:18:00 0:18:05 These are sort of the two major kinds of protection that we 0:18:09.187 have, and we need to make sure that our protocols we use sort 0:18:13.518 of provide these kinds of guarantees. 0:18:16.117 Now what I want to do is talk a little bit about authorization. 0:18:20.593 We've seen authentication and a bit of confidentiality and we've 0:18:25.141 talked about authorization, but I want to talk about it in 0:18:29.256 the context of an example. Because you guys know something 0:18:33.963 about authorization techniques. You guys have all seen, 0:18:37.384 for example, passwords for logging into a 0:18:39.918 system. There is no big surprise about 0:18:42.262 how passwords work. There is some list of 0:18:44.796 passwords. And, when a user tries to log 0:18:47.266 in, the system checks to see if the person is in the list. 0:18:50.877 They typically take the hash of the password that is typed in 0:18:54.678 and check to see if the hash of the password is in the password 0:18:58.606 list. This is described in detail in 0:19:01.677 the text and we will go over it a little bit today, 0:19:04.473 but what I want to do is sort of talk about how we fit all 0:19:07.661 three of these things, authentication, 0:19:09.73 authorization and confidentiality together into 0:19:12.302 one system as we talk about authentication. 0:19:14.651 We are going to use an example. 0:19:17 0:19:22 Which is the Web. Suppose that we were in a 0:19:26.846 situation where we have some browser B communicating with 0:19:33.307 some Web server W over a secure communication channel. 0:19:40 0:19:45 And, if you like, you can think of the secure 0:19:47.652 communication channel as having been established by a protocol 0:19:51.329 like the one I've shown here. In practice on the Web, 0:19:54.464 there is a common protocol that is used to establish the secure 0:19:58.202 communication channel called SSL, secure sockets layer. 0:20:02 This channel has been authenticated and is 0:20:07.72 confidential. It is typically the case, 0:20:13.023 for example, that there may be some 0:20:17.767 certificate authority, for example, 0:20:22.511 which B has used to discover keys for W, to discover a 0:20:29.906 private key to use to talk to the Web server, 0:20:36.046 W. There is a question, 0:20:41.826 though, that we haven't really got at which is how does W know 0:20:53.495 that B is authorized to access? 0:21:00 0:21:05 And we hinted at this on the first lecture about security, 0:21:09.819 but this is the question that we want to try and address today 0:21:14.976 in a little more detail. The issue is that once this 0:21:19.288 protocol is established, these two guys have exchanged a 0:21:23.938 key with each other, this kAB or kBW. 0:21:26.982 And this is a shared key that these guys can communicate with 0:21:32.055 each other. But basically all that W knows 0:21:36.437 about B is that this is somebody who has this key. 0:21:40.418 This is the same B who initiated this connection with 0:21:44.643 me. W may not have actually checked 0:21:47.406 to go ahead and see what it is on the server, 0:21:50.981 what services on WB has access to. 0:21:53.662 So W needs to go ahead and figure out what it is that B can 0:21:58.375 access. And these two sort of check all 0:22:02.753 the accesses that B tries to make on W to make sure that it 0:22:07.906 is authorized to do so. If you remember back to the 0:22:12.347 very first lecture, we talked about how 0:22:15.723 authentication and authorization work. 0:22:19.01 We sort of said there are two steps. 0:22:22.119 Or, in this case, we are going to actually talk 0:22:26.205 about three steps. We have three steps or let's 0:22:30.291 call it three authorization functions. 0:22:35 0:22:40 And these functions actually sort of share something in 0:22:44.821 common with authentication because authenticating and 0:22:49.464 authorizing are sort of intertwined together. 0:22:53.392 You will see what I mean. The first step is we need some 0:22:58.303 rendezvous. When we talk about this, 0:23:01.674 this is the way that two principles initially set up the 0:23:05.511 access rights to the particular server. 0:23:08.162 For example, this is you going to your 0:23:10.744 system administrator and telling your system administrator that 0:23:15.069 you would like an account. He creates an account, 0:23:18.418 he creates a home directly for you and then you have the rights 0:23:22.744 to access any of the files that are in your home directly. 0:23:26.72 So think of rendezvous as setup. 0:23:30 Account creation, you can log onto Amazon dot com 0:23:33.755 and create an account, for example. 0:23:36.415 Then there is some other step which is this verification step. 0:23:41.188 And verification is simply making sure that when you 0:23:45.178 reconnect to the system you are allowed to connect to the things 0:23:50.107 that you want to

connect to. Typically, in an authorization 0:23:54.645 system, we talk about this thing mediating your communication 0:23:59.339 with the Web server. If I log onto Amazon dot com 0:24:04.177 and say that I would like to log in as a particular user, 0:24:08.533 this verification step is going to make sure that I have the 0:24:13.122 appropriate credentials to log on as that user. 0:24:16.7 The final thing I might want to do is to revoke. 0:24:20.355 I might want to simply remove a user from the system or make it 0:24:25.177 so that the user is no longer a part of the system. 0:24:30 0:24:35 There are two widely sort of used approaches for 0:24:41.317 authentication that sort of do different things at these 0:24:48.709 different steps. The two approaches are called 0:24:54.758 lists and tickets. Authorization. 0:25:00 We have these two approaches. The first one we are going to 0:25:06.468 call lists and the second one is called tickets. 0:25:11.71 And we have these three steps. We have our setup step, 0:25:17.62 we have our mediation step and we have our revocation step. 0:25:24.089 Lists are something you may be familiar with. 0:25:30 One way that we might authorize whether or not a user is allowed 0:25:34.295 to access a system is to check some list of all the users who 0:25:38.386 are allowed access, and we may check that user's 0:25:41.59 credentials. For example, 0:25:43.227 if you go to a party and that party is invite only and 0:25:46.84 requires you to show your MIT ID at the door, you show up at the 0:25:51.136 door, you show them your MIT ID and then that lets you in. 0:25:55.022 And now you've been sort of authorized to access the party. 0:26:00 The other approach is a ticket-based approach. 0:26:02.689 This is an approach where instead of having a list of 0:26:05.796 people and checking credentials whenever the person wants 0:26:09.143 access, instead you have some ticket which lets anybody who 0:26:12.609 has that ticket have access. This is a party is invitation 0:26:16.015 only. You get an invitation in the 0:26:17.988 mail. And now anybody who has that 0:26:19.96 invitation can go ahead and go to the party. 0:26:22.529 You guys are all familiar with systems that work like that, 0:26:25.996 baseball games, carnival games or whatever it 0:26:28.625 is. You get tickets and you use 0:26:31.38 those tickets to get access to something. 0:26:33.424 And anybody who has those tickets can use them. 0:26:35.776 There is no checking of your sort of credentials every time 0:26:38.741 you try to use them. As we're going to see, 0:26:40.888 and we will talk about how this works on the Web, 0:26:43.341 it is often the case that you use one of these list-based 0:26:46.204 authentication systems to decide who we should issue tickets to. 0:26:49.424 And then you give a bunch of people tickets and anybody can 0:26:52.389 do whatever they want with those tickets. 0:26:54.434 Let's talk about sort of the properties of these things and a 0:26:57.501 little bit about how they work in the context of a computer 0:27:00.466 system. The setup process and list is 0:27:03.938 obviously you add someone to a list. 0:27:06.277 Passwords typically are done with list-based systems. 0:27:09.752 The idea is when you add somebody to a password list you 0:27:13.428 input their name or their account name and a cryptographic 0:27:17.237 hash of the password that they typed in. 0:27:19.844 Usually, you want them to type the password in, 0:27:22.918 in some secure fashion. You put the cryptographic hash 0:27:26.46 of that password there. And then, in order to mediate, 0:27:30.852 in order to actually verify, for example, 0:27:33.322 that the password is going to be correct, you are going to 0:27:36.842 search the list and you are going to verify that the 0:27:39.992 password that they present, in fact, hashes to the stored 0:27:43.45 hash in the password file. You are going to search the 0:27:46.723 list and you're going to also do this sort of check credentials 0:27:50.551 step and make sure the password is what it was supposed to be. 0:27:54.318 It is relatively easy then, in this environment, 0:27:57.221 to revoke somebody's access. You can remove them from list. 0:28:03.448 And then, in that case, well, they won't be able to use 0:28:09.655 the system anymore. Tickets. 0:28:12.758 The idea in a ticket-based system is that you are going to 0:28:19.31 generate a ticket. And usually what that means in 0:28:24.827 a computer system, or the simplest way to generate 0:28:30.459 a ticket in a computer system is to make a sort of hard to guess 0:28:37.701 number. For example, 0:28:40.645 this might be a hash of something or it might just be a 0:28:43.548 big random number. You want it to be something 0:28:45.967 that if somebody picks another random number out of thin air 0:28:49.139 they don't have a very high probability of guessing a number 0:28:52.311 that actually allows them to access the system, 0:28:54.784 but anybody who has this ticket should be able to access the 0:28:57.956 system relatively easily. Because these tickets are just, 0:29:02.004 for example, a big random number, 0:29:04.073 users typically should feel free to share these, 0:29:07.112 might be able to exchange these tickets with each other. 0:29:10.668 They should be able to share them with other people and use 0:29:14.418 them for access. Whereas, users typically are 0:29:17.262 not going to share their passwords with other people. 0:29:20.625 Tickets are a way that a user can handoff sort of the 0:29:23.987 authority, the right to access a system to some other user, 0:29:27.737 they can delegate. Now, to mediate we are just 0:29:32.05 going to look up, this is just going to be a 0:29:35.316 table lookup. We just need to make sure that 0:29:38.582 this is a valid number that was, in fact, given to us. 0:29:42.607 And, typically, we don't have to go and check 0:29:45.949 credentials. The user doesn't have to supply 0:29:49.215 a password when they supply us with a ticket. 0:29:52.556 And then, finally, in order to revoke, 0:29:55.367 well, we might be able to invalidate a ticket. 0:30:00 You might be able to say, for example, 0:30:02.07 remove her from the table. That works fine if we're OK 0:30:05.037 with getting rid of an entire ticket. 0:30:07.052 But suppose, for example, 0:30:08.395 that I want to revoke the right of a single user to access the 0:30:11.809 system. That is very hard to do in a 0:30:13.768 ticket-based system. Suppose that you guys decide 0:30:16.455 that you no longer want me looking at some discussion board 0:30:19.701 about 6.033 that you've set up. And suppose because you're 0:30:22.891 writing bad things about me and you say, well, 0:30:25.41 we don't want the professor to be able to access this anymore. 0:30:30 So you try and revoke my access. 0:30:31.848 If you've been using a ticket-based system that is 0:30:34.771 going to be very hard unless you invalidate that ticket. 0:30:38.051 And there are many people who are sharing the same ticket. 0:30:41.451 That's going to be hard to deny me access because you're going 0:30:45.089 to have to revoke that ticket and then go to everybody else 0:30:48.548 who might be sharing that ticket and issue them a new one. 0:30:51.948 Whereas, in this sort of list-based system you could have 0:30:55.288 just removed my account. This has an analogy in the real 0:30:58.568 world. For example, 0:31:00.641 door locks are sort of like a ticket-based system. 0:31:03.262

You generate a physical key for something. 0:31:05.454 You could give that out to many people, and then many people can 0:31:08.823 have access to your house. The problem is if you want to 0:31:11.764 revoke access to a single person then you have to sort of go and 0:31:15.133 either collect the keys from everybody, which may not be 0:31:18.074 feasible, or change the door locks which denies everybody 0:31:21.069 access until you reissue keys to the people who need them. 0:31:24.117 If you like, you can think of a ticket-based 0:31:26.417 system sort of like being door locks in a house. 0:31:30 0:31:35 Oftentimes lists are more formally called ACLs or access 0:31:39.969 control lists. You will see this term used in 0:31:43.945 the text, in fact, as well. 0:31:46.295 And tickets are sometimes called capabilities. 0:31:50.361 Capabilities are sort of the right to access some resource on 0:31:55.783 the computer that can be delegated from one user to the 0:32:00.662 other or one principle to the other. 0:32:05 And, in practice, what we see both, 0:32:07.116 as I said, in the real world and in computer systems is that 0:32:10.788 we are often going to use this list-based mechanisms to decide 0:32:14.585 who to hand tickets out to. In the case of, 0:32:17.199 OK, who are we going to invite to our party, 0:32:19.875 who are we going to send invitations to in the mail, 0:32:23.049 well, you might have some list that you go ahead and assemble 0:32:26.784 based on either people who you know to be MIT students or 0:32:30.269 people who meet some particular set of eligibility criteria. 0:32:35 That is the sort of process of checking a list. 0:32:38.074 And then, once you've assembled the list, you are going to 0:32:41.885 generate these tickets and send them out, and then anybody can 0:32:45.962 use those tickets. This same analogy sort of holds 0:32:49.237 true in the Internet, so let's look at how we might 0:32:52.58 build up an authorization system on top of our secure channel 0:32:56.59 between our browser and our Web server. 0:33:00 We have our B talking over our secure channel of our Web 0:33:04.23 server. And the idea is going to be as 0:33:07.076 follows. This Web server has some 0:33:09.538 service that the user wants to try and access, 0:33:13 and we are going to have this guard. 0:33:15.692 Remember, we talked about this guard model before. 0:33:19.461 There is going to be some guard that sits in front of this 0:33:23.846 service that is in charge of authorizing B to access this 0:33:28.153 service. And then there is some 0:33:31.803 authorization module that runs on B. 0:33:34.672 The idea is that B is going to initiate a connection and then 0:33:39.59 guard is going to send a message to the authorization module on B 0:33:44.836 saying who are you, what are your credentials to 0:33:48.688 access this system? This is going to be a form of 0:33:52.622 list-based access. 0:33:55 0:34:00 Now, this module on B is going to go ahead and send a message 0:34:04.74 back that says my name is, whatever my name is, 0:34:08.374 as well as some key, some credential like, 0:34:11.613 for example, a password that is their 0:34:14.457 password for accessing this system. 0:34:17.143 Notice that these two guys do not need to worry about sort of 0:34:21.883 protecting this channel, as far as the rest of the 0:34:25.754 Internet is already concerned, because they have already 0:34:30.099 established this secure channel upon which they can do this 0:34:34.681 exchange of information. So they are hopefully not 0:34:39.244 worried about other people overhearing this message. 0:34:41.783 They may not need to protect this information that is being 0:34:44.67 transmitted in the same way because they have already got 0:34:47.458 the secure communication channel that they have established at 0:34:50.495 the lower levels of this thing. All of this communication is 0:34:53.432 going over this secure communication channel between 0:34:55.971 these two guys. This is a layer that is built 0:34:58.162 on top of this secure communication layer. 0:35:01 Now what happens is that what the guard is going to do is look 0:35:04.971 up this name and password, for example, 0:35:07.444 in some access control list and use that to determine whether or 0:35:11.546 not this person has the rights to access and what they have the 0:35:15.582 rights to access to. And he is going to generate for 0:35:18.902 this person a set of tickets that represent the sort of 0:35:22.417 services that this person can access on this system. 0:35:25.737 He is going to have some table of tickets, and this table is 0:35:29.578 going to have the ticket number and the resource, 0:35:32.703 for example, that the user is allowed to 0:35:35.242 access. So he is going to send back 0:35:39.667 some ticket number authNo back to the user. 0:35:43.818 So we are going to add authNo. And then maybe this gives him 0:35:49.648 the right to access some account, B's account 0:35:53.996 information. Now, with this authorization 0:35:57.948 information, this is the ticket. Now every time that B wants to 0:36:04.718 go ahead and access his account, for example, 0:36:08.705 he can read account B and then he just passes the ticket. 0:36:13.779 And he can use this ticket over and over and over again to 0:36:18.944 access the account for as long as the ticket is valid. 0:36:23.747 Often times there will be a timeout associated with tickets. 0:36:30 But as long as the ticket is valid he can go ahead and supply 0:36:34.519 this ticket in order to be able to access the resource that the 0:36:39.189 ticket gives him access to. And he doesn't have to sort of 0:36:43.483 re-authenticate himself. He doesn't have to represent 0:36:47.4 his credentials every time he wants to access this resource on 0:36:51.994 his account on the Web server. Are the HKN people here? 0:36:56.062 Yes. OK. 0:36:56.589 We need to do HKN. Why don't you just give me one 0:37:01.506 more minute to wrap up and then we can go ahead and do the HKN 0:37:06.603 review. This protocol, 0:37:08.358 I mentioned that this is sort of like SSL, or the SSL works in 0:37:13.455 sort of this way. On the Internet you may have 0:37:17.215 heard about cookies. Cookies on the Web are 0:37:20.725 essentially a kind of ticket. The idea with most cookie-based 0:37:25.738 systems is you log in with a password. 0:37:30 And then, when you log in with that password, 0:37:33.196 the system issues you a cookie. And then when you want to 0:37:37.264 re-access the system you simply supply the cookie. 0:37:40.824 If you were to look at the contents of a prototypical 0:37:44.602 cookie, it would typically have something like, 0:37:47.944 for example, the user's name and some 0:37:50.559 timeout, a valid period for the cookie, as well as a hash of the 0:37:55.136 user's name, that timeout and some, for example, 0:37:58.551 random number which is only known over here at the service. 0:38:04 The service is going to protect this random number and is going 0:38:07.126 to hash it together with the user's ID and this timeout. 0:38:09.9 And so now, when this cookie is supplied, it is going to be hard 0:38:13.078 for users to generate fake cookies, but these cookies are 0:38:15.902 going to allow the user to go ahead and access the services on 0:38:18.979 W without having to re-authenticate every time. 0:38:21.299 And you can, in fact, in some cookie systems 0:38:23.467 share those cookies. You could copy the cookie from 0:38:25.989 one

fact, in some cookie systems clients can share these cookies. You could copy the cookie from client one browser to another, and you might be able to reuse 0:38:28.662 that cookie to access the system for as long as the cookie is 0:38:31.688 valid. Cookies typically timeout after 0:38:35.191 an hour or two, and so then you would have to 0:38:38.301 re-authenticate with the system. That is all I wanted to really 0:38:42.683 say about authentication. What we are going to do is talk 0:38:46.641 about some sort of advanced protocols next time. 0:38:49.963 I want to remind you guys that there is no class on this 0:38:53.85 Wednesday so don't come here. The next class is next Monday. 0:38:58.02 And that is it. Good luck finishing your DP2 0:39:01.948 and have fun doing HKN reviews.