Network Security

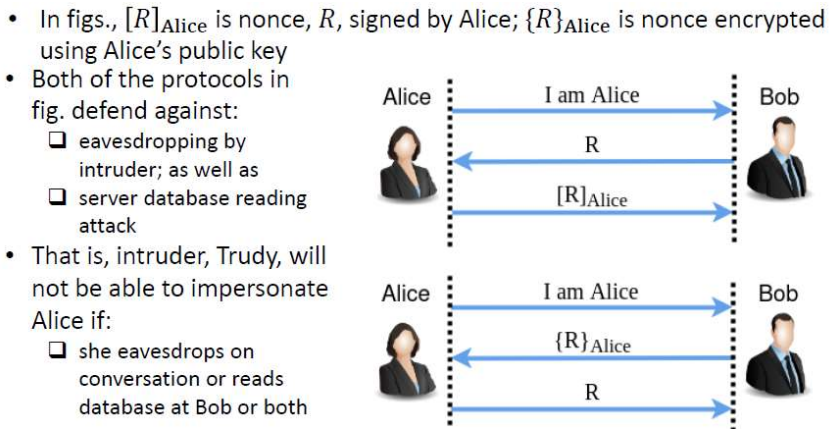
Professor Gaurav S. Kasbekar
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Week - 04

Lecture - 22

Authentication: Part 3

Hello, recall that in the previous two lectures, we discussed one-way authentication and mutual authentication. We will now discuss some more protocols for one-way authentication. So, recall the protocols that we discussed for authentication using public keys. Suppose Alice has a public key-private key pair, and the public key of Alice is known to Bob. We can use either of these protocols to authenticate Alice to Bob. In these figures, this notation is the nonce, R, signed by Alice, and this notation is the nonce encrypted using Alice's public key.



So, in the first protocol above, Alice sends "I am Alice" to Bob to initiate the authentication process. Bob sends R, which is a challenge, and Alice signs it using her private key. This is the signed result, and Alice sends the signed result to Bob. Bob applies the public key of Alice to this to check whether he gets back the nonce, R. In this second protocol, Alice initiates the authentication by sending, "I am Alice".

Bob selects the nonce R, encrypts it using Alice's public key, and sends the result to Alice. Alice decrypts it using her private key and sends the recovered nonce to Bob. Bob checks whether that is indeed the nonce he had selected. So, notice that both of these protocols

defend against eavesdropping by an intruder. If an eavesdropper is listening to the communication channel between Alice and Bob, these protocols defend against such an eavesdropper. These protocols also defend against the server database reading attack, where some attacker reads the database at the server Bob.

So, these protocols also defend against the case where an intruder is eavesdropping on the communication channel as well as has read the database at the server Bob. So, these protocols defend simultaneously against both eavesdropping by intruders as well as the server database reading attack. So, that's because no confidential information is sent over the channel; hence, these protocols defend against eavesdropping. And also, at the server, Bob, only Alice's public key is stored. No secret information is stored at the server, Bob.

So, hence, they defend against the server database reading attack. So, in other words, an intruder, Trudy, will not be able to impersonate Alice if she eavesdrops on the conversation or reads the database at Bob or both. Now, we want to authenticate Alice to Bob while defending against both eavesdropping and the server database reading attack. We discussed two protocols on the previous slide which achieve this. They authenticate Alice to Bob while defending against both eavesdropping and the server database reading attack.

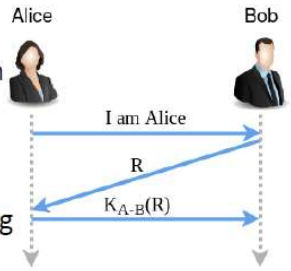
But can this be done without using public key cryptography? We know that public key cryptography is computationally expensive. So, can we achieve the same objectives without using public key cryptography? We'll show that the answer is yes. But before that, first we show that it is easy to defend against any one of the above two attacks, eavesdropping and server database reading attack, if you do not defend against the other attack.

So, it is easy to defend against eavesdropping if we don't defend against the server database reading attack and vice versa. So, let's start with defense against eavesdropping. Assume that the server database reading attack does not take place. So, can we design a protocol which defends against eavesdropping? So, we have already discussed the protocol which defends against eavesdropping.

Suppose Alice and Bob have a shared symmetric key $K_{Alice\text{-}Bob}$. This protocol, ap4.0, which we discussed, defends against eavesdropping. So, if an eavesdropper intercepts the entire communication between Alice and Bob, they won't get the secret $K_{A\text{-}B}$. We also discussed a variant of this protocol in which Alice sends $H(K_{A\text{-}B}, R)$ to Bob, where H is a cryptographic hash function. Alice sends this instead of $K_{A\text{-}B}(R)$. So, this variation also defends against eavesdropping.

But notice that if these protocols are used and the intruder reads the database at the server Bob, then the intruder will get the secret key $K_{A\text{-}B}$, and these protocols will be compromised. So, this protocol, ap4.0, and its variation, in which cryptographic hash functions are used, these protocols defend against eavesdropping only. They don't defend against the server database reading attack. So, now, let's try to defend against the server database reading attack, assuming that no eavesdropping takes place. So, consider a protocol in which Alice selects a password, say p, and the server Bob only stores the hash of p; let's call it H(p). So, notice that Bob does not know the password; Bob only knows the hash of the password, H(p).

- Defence against eavesdropping:
  - ❑ Suppose Alice and Bob have a shared symmetric key $K_{Alice-Bob}$
  - ❑ Protocol ap4.0 (shown in fig.) and its variation in which Alice sends $H(K_{Alice-Bob}, R)$ to Bob instead of $K_{Alice-Bob}(R)$ defend against eavesdropping
  - ❑ Do not defend against server database reading attack



To authenticate, Alice sends the message "I am Alice, p" to Bob, and then Bob finds the hash value of p and checks whether it equals the stored hash value, that is, H(p). If yes, then the authentication is successful. This defends against the server database reading attack, but not against eavesdropping. The reason is that if an intruder reads the database at the server Bob, then they will get H(p), but H(p) is not enough to authenticate as Alice to Bob, because to authenticate as Alice to Bob, one requires sending a password p to Bob, and it is computationally infeasible to get the password p from the hash value H(p). So, someone who reads the server database gets H(p), from which it is computationally infeasible to get the password p; hence, this protocol defends against the server database reading attack. But it does not defend against eavesdropping.

- ❑ Alice selects a password, say p
- ❑ Bob stores hash of p, say $H(p)$

If there is an eavesdropper who intercepts all the messages sent on the communication channel between Alice and Bob, then they will get the password p, and they can subsequently authenticate as Alice to Bob. So, this protocol does not defend against

eavesdropping. Next, we study a protocol which defends simultaneously against both eavesdropping and the server database reading attack without using any public key cryptography. This protocol is called Lamport's hash protocol since it was invented by the well-known scientist Leslie Lamport. So, Lamport's hash works as follows.

Alice, who is a human, remembers a password. Bob is a server to which Alice would like to authenticate. Bob has a database where it stores the following quantities for each user. The username of the user, then n, is an integer which decrements each time Bob authenticates the user. n is the number of times that the protocol can authenticate Alice to Bob without requiring a reconfiguration of the protocol.

So, once the protocol is configured, Alice can authenticate herself n times to Bob. And for authenticating n times, there is some reconfiguration required, as we'll see. Then Bob stores for each user also $h^n(password)$, where password is Alice's password. By $h^n(password)$, we mean the hash function applied n times to the password. That is hash(hash(...(hash(password)...)), where the hash is applied n times.

So, Bob stores these quantities for each user: Alice, the username; n, which is an integer; and a $hash^n(password)$. A typical value of n is something like 1,000. Before Alice and Bob can use the authentication protocol, the password database entry at Bob for Alice is configured as follows. Alice uses a password. Her workstation computes a $hash^n(password)$, where n is some large value like 1,000.

- ❑ $\text{hash}^n(password)$, i.e., $\text{hash}(\text{hash}(\ldots(\text{hash}(password))\ldots))$
- Before Alice and Bob can use the authentication protocol, the password database entry at Bob for Alice is configured as follows:
  - ❑ Alice chooses a password, her workstation computes $\text{hash}^n(password)$, where n is some large value like 1000
  - ❑ the values <Alice, n, $\text{hash}^n(password)$> are sent to Bob

Then these values, $\langle Alice,\ n, hash^n(password)\rangle$, this triplet is sent to Bob. For this protocol to work correctly, some ways required to securely communicate this triplet, $\langle Alice,\ n, hash^n(password)\rangle$, from Alice to Bob, but this is required only at the time of configuration, that is, once every n authentication attempts by Alice. So, for every authentication attempt, this is not required. This is a drawback of Lamport's hash scheme. We require some way to securely communicate this information: $\langle Alice, n, hash^n(password)\rangle$ from Alice to Bob.

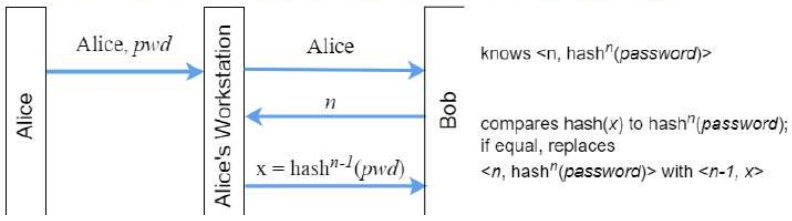
But this is only a minor drawback because this needs to be securely communicated once every n attempts, where n can be large, like 1,000 or so. So, this is the configuration step. Now, suppose the password database entry at Bob for Alice has been configured. That is the information that we discussed on the previous slide, which has been stored at Bob's database for the user Alice. Now, Alice wants to authenticate to Bob.

The authentication protocol is shown in this figure here. This is Alice's workstation. And this is the server, Bob. In the first step, Alice types her username and password at her workstation. That's shown here.

Alice types her username, Alice, and the password at the workstation. Then, Alice's workstation sends the username, Alice, to Bob to initiate the authentication, and Bob knows which user wants to authenticate. Now, we have seen that during configuration Bob has the triplet, $\langle Alice,\ n, hash^n(password)\rangle$ stored for Alice. Now, Bob sends n to Alice's workstation, which is the integer that is a part of this information. Now, Alice's workstation computes this quantity $x = hash^{n-1}(password)$, and sends it to Bob.

- ❑ Alice's workstation computes $x = \text{hash}^{n-1}(password)$ and sends it to Bob
- ❑ Bob takes x, computes its hash, $\text{hash}(x)$, and compares result to $\text{hash}^n(password)$
- ❑ If the two match, Bob considers response valid and Alice's authentication is successful
- ❑ Then Bob replaces $<n, \text{hash}^n(password)>$ with $<n-1, x>$

- When value of n gets to 1:
  - ❑ Password database entry at Bob for Alice has to be reconfigured



So, this x is obtained by applying the hash function n-1 times to the password. So, notice that Alice has already typed her password at the workstation, so the workstation knows the password and is able to compute $hash^{n-1}(password)$. So, Alice's workstation sends x, which is $hash^{n-1}(password)$, to Bob. Now, to verify whether on the other side there is indeed Alice or not, Bob takes the quantity x, computes its hash(x), and compares the result to $hash^n(password)$. So, if x is indeed equal to $hash^{n-1}(password)$, then hash(x) will be equal to $hash^n(password)$, and the authentication will be successful.

So, in order to authenticate themselves as Alice to Bob, one would need to know the password from which one can compute $hash^{n-1}(password)$. If there's a match at Bob, then Bob considers the response valid, and Alice's authentication is successful. Then Bob replaces $\langle n, hash^n(password)\rangle$ for Alice's entry with $\langle n-1, x\rangle$. So, this integer is decremented by one, and now $hash^n(password)$ is replaced with $hash^{n-1}(password)$. That is the same as x. So, now after this replacement, another authentication can take place.

So, when the value of n gets to 1, x would become $hash^0(password)$, that is, the password itself. It is not secure to send the password over the channel because there may be eavesdropping. So, hence, we require reconfiguration. In this case, the password database entry at Bob for Alice has to be reconfigured. So, the reconfiguration takes place, and then for another n attempts, authentication can take place.

So, let's analyze this protocol now. So, does this scheme defend against eavesdropping by an intruder? Suppose there is an intruder who intercepts all these messages exchanged on the channel between Alice and Bob. Such an intruder, Trudy, will obtain the values of n and $hash^{n-1}(password)$. But suppose Trudy tries to authenticate as Alice to Bob subsequently using this stolen information.

If she attempts to authenticate as Alice, Bob will send the value n-1 to her because, at the end of the first successful authentication, Bob had replaced n with n-1. So, now Bob sends the integer n-1 to Trudy. And Trudy needs to send $hash^{n-2}(password)$. But Trudy is not able to compute $hash^{n-2}(password)$ from $hash^{n-1}(password)$ because of the one-way property of the hash function. So, hence, this protocol defends against eavesdropping by an intruder.

- ❑ Yes; an eavesdropping intruder, Trudy, will obtain the values of n and $\text{hash}^{n-1}(password)$
- ❑ But if she attempts to authenticate as Alice, Bob will send $n-1$ to her and Trudy does not know $\text{hash}^{n-2}(password)$

- Does this scheme defend against the server database reading attack?
  - ❑ Yes; if Trudy reads database at Bob, she will obtain $<n, \text{hash}^n(password)>$

Does this scheme defend against the server database reading attack? Suppose an intruder manages to read the database at Bob; will the intruder be able to authenticate as Alice to Bob? This scheme does defend against the server database reading attack. If Trudy reads the database at Bob, she will obtain the values n and the $hash^n(password)$ for the user

Alice. But for authenticating as Alice to Bob, the intruder needs to know $hash^{n-1}(password)$.

$hash^{n-1}(password)$ cannot be computed from $hash^{n}(password)$ because of the one-way property of the cryptographic hash function. This is not known to Trudy; hence, Trudy is not able to authenticate as Alice to Bob. So, hence, this scheme defends against the server database reading attack. Now, suppose Alice uses the same password to log into multiple servers. Then, the following attack is possible.

An eavesdropping intruder who obtains the values of n and $hash^{n-1}(password)$ when Alice logs into one server by eavesdropping on the channel. Such an intruder can use these values to authenticate as Alice at another server. So, an intruder captures all these values; Alice, n, and $hash^{n-1}(password)$, and tries to authenticate as Alice at another server. If the other server is also initialized with the same n, then the other server will send n, and the intruder, Trudy, knows $hash^{n-1}(password)$, which is the correct response. So, how can we defend against this attack?

So, we need to add something which is specific to the server Bob to this authentication process. So, one way to do this is as follows. During configuration, $\langle Alice, n, hash^{n}(password|servername)\rangle$, that is, Bob in this case, is communicated from Alice to the server. So, this triplet that is stored at the server is different for every server because it has a server name concatenated before computation of hash n. So, this is communicated from Alice to Bob instead of communicating $\langle Alice, n, hash^{n}(password)\rangle$.

- Defence against this attack:
  - ❑ During configuration, <Alice, n, hash$^{n}(password|servername)$> is communicated from Alice to the server
  - ❑ Authentication exchange: server sends n to Alice; Alice responds with hash$^{n-1}(password|servername)$

Hence, this triplet is different for different servers, and the correct response to the challenge n is now $x = hash^{n-1}(password)$ concatenated with the server name, Bob. So, hence, by overhearing the conversation that takes place between Alice and one server, an intruder is not able to authenticate to another server because the response expected is different since the server name is different. The authentication exchange is this. The server sends n to Alice, and Alice responds with $hash^{n-1}(password)$ concatenated with the server name. So, this protocol defends against the attack that we just described.

Now, suppose an intruder, Trudy, impersonates Bob's network address and waits for Alice to log in. So, when Alice initiates authentication, a request for authentication that is Alice reaches Trudy instead of Bob. When Alice attempts to log into Bob, Trudy intercepts the request and sends a small value of n to Alice, say n = 50. Then, suppose the actual n=1000. In this case, Alice responds with $hash^{49}(password)$ because $hash^{n-1}(password)$ is $hash^{49}(password)$.

- When Alice attempts to log in to Bob:
  - Trudy sends a small value of n to Alice, say $n = 50$
  - suppose actual $n = 1000$
- When Alice responds with $\text{hash}^{49}(password)$, Trudy has obtained enough information to impersonate herself as Alice to Bob ≈ 950 times
  - called "*small n attack*"

So, once Alice responds with this value, Trudy has obtained enough information to impersonate herself as Alice to Bob approximately 950 times. Because Trudy has obtained the value of $hash^{49}(password)$, then Trudy can send a message, Alice, to the actual server Bob, and when Bob sends the true value of n, which is 1000, Trudy has to send $hash^{999}(password)$, which Trudy is able to compute because $hash^{49}(password)$ is known. So, by just applying the hash function to this value, 999-49 times, that is, 950 times, Trudy is able to compute $hash^{999}(password)$. So, that is the first authentication. Then Trudy can similarly authenticate one more time as Alice.

At that point, Trudy needs to know $hash^{998}(password)$, and so on and so forth. So, Trudy can impersonate herself as Alice to Bob approximately 950 times. So, this is called a small n attack, where Trudy sends a small value of n to Alice and extracts the value of $hash^{49}(password)$, due to which Trudy is able to impersonate herself as Alice to Bob a large number of times. So, what is the defense against this attack? We can maintain a counter at Alice, which keeps track of the correct value of n. So, in this example, Alice knows that the correct value of n=1,000, so if Alice receives some value like 50, then Alice knows that the small n attack is ongoing.

So, Alice will not respond with the correct response. So, by maintaining a counter at Alice which keeps track of the correct value of n, we can defend against this attack. This concludes our discussion of Lamport's hash protocol. Thank you.