Network Security

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

Week - 04
Lecture - 23
Authentication: Part 4

Hello, in the previous lecture, we discussed Lamport's hash protocol, which is a protocol that can be used for authentication. Now, we will discuss how communication can securely take place after authentication has occurred. And we will also discuss some other attacks related to authentication. So, suppose Alice and Bob mutually authenticate using one of the protocols that we discussed. Now, what about the rest of the conversation?

For example, Alice might be a user who connects to Bob, who is an email server. Then, after authentication, Alice will check emails in her inbox, send messages, and so on. So, if this rest of the conversation is transmitted without any security mechanisms, then the following attacks can take place. Trudy, who is an intruder on the path between Alice and Bob, can read and/or modify the contents of packets sent by Alice and/or Bob. Trudy can replay packets sent between Alice and Bob, and so on and so forth.

So, such attacks are possible if the communication after authentication is done without any security mechanisms. We need to provide encryption and message integrity for the data that follows the authentication exchange. Now, recall that encryption and message integrity require secret keys. So, for symmetric key encryption, we know that the sender and receiver have to agree on a secret symmetric key, K_S, and for message integrity, if we use a MAC, that is a message authentication code, then the sender and receiver have to know the secret s, which is used to compute H(m,s) for the message m. So encryption and message integrity require keys. What should Alice and Bob use for keys?

One possibility is this: Alice encrypts all the data she sends to Bob using Bob's public key. This encryption is done to achieve confidentiality. Alice also signs the information that she sends using her own private key. This is done for message integrity. So, Alice creates a digital signature, which achieves message integrity as we have discussed in an earlier lecture.

Similarly, in the other direction, for confidentiality, Bob encrypts all the data he sends using Alice's public key, and for message integrity, he signs the information using his own private key. So, this assumes that Alice and Bob have private and public key pairs. Besides, there is a shortcoming of this approach: public key operations are computationally expensive. So, after authentication, Alice and Bob may communicate, and they may exchange a lot of messages. So, all these messages have to be encrypted, and their integrity has to be provided using public key techniques.

So, this is very computationally expensive. That's the shortcoming of this approach. So, it's more practical to have Alice and Bob agree on a secret symmetric key and protect the rest of the conversation using that key. So, in this approach, during authentication, Alice and Bob also agree upon secret keys, which they'll use to protect the rest of the conversation—that is, to achieve confidentiality and message integrity for the rest of the conversation. Hence, it is useful if the authentication protocol helps Alice and Bob agree on a secret key in addition to providing mutual authentication.

So, one possibility is the following. When Alice and Bob use a shared symmetric key to authenticate each other, they continue to use that same key to protect the rest of the conversation. For example, consider protocol ap4.0. Alice and Bob know a secret key, K_{A-B}. We also discussed the mutual authentication version of ap4.0, where the authentication is repeated in each direction.

So, in that case, Alice and Bob know the shared symmetric key K_{A-B}. So, they can use the same key to protect the rest of the conversation. But it is more secure to generate a separate session key, which is different for different sessions. So, the key K_{A-B} is used for authentication, but after authentication, they agree upon a separate session key, which is different for different sessions. And this session key is used for encryption and message integrity of the messages that are sent after authentication.

The reasons for this are as follows. Consider an intruder who is eavesdropping on the conversations between Alice and Bob and is trying to find the long-term key, K_{A-B}, which is used for authentication. Then, if the same key is also used for encryption and message integrity, then the intruder is able to collect a lot of encrypted data, all encrypted using the same key K_{A-B}. So, if an intruder is trying to break the cipher and find a key, if they manage to collect a lot of encrypted data, then they are more likely to succeed. So, the more the ciphertext that is collected corresponding to a particular secret key, the easier it is to break the cipher and obtain the secret key.

So, this is one reason for using a different session key. For using a different key for encryption and message integrity, this key is different from that used for authentication. And the key is different for every session. So, it is only temporarily used for a particular session, and then next time Alice and Bob communicate, a different key is used. Another reason is that it may be possible for an intruder to record messages from a previous Alice-Bob conversation and inject them into a current conversation.

So, Alice and Bob might communicate on one day, then end the session, and then communicate again on another day. Within a session, we'll see that sequence numbers are used to prevent an intruder from replaying, deleting, or reordering messages. For example, the first packet sent in a session might be numbered 1, the second packet might be numbered 2, and so on. So, such sequence numbers are added to packets, as we'll see. But there may be nothing in packets to distinguish packets transmitted during an old conversation from packets transmitted during the current conversation.

For example, the protocol may always start with sequence number 1. If the key K_{A-B} is used for encryption and message integrity in every session, then an intruder can take a packet with sequence number n from an old session and play it in place of the packet with sequence number n in the current session. So, this is only possible for an intruder if the same key is used for every session. So, this is another reason for using a different session key for different sessions. So for these reasons, during authentication, Alice and Bob agree on a secret session key, which is used for that particular session for achieving confidentiality and message integrity.

And in the following session, they agree on a different key. So, the session key changes from session to session. Due to these reasons, as part of an authentication protocol, after Alice and Bob authenticate each other (or after a one-way authentication), they need to agree upon a session key. So, the question is, how do we select the session key? We'll discuss how to establish a session key for each of the following cases.

1) Alice and Bob share a secret symmetric key $K_{Alice-Bob}$ (used for authentication)

The first case is when Alice and Bob share a secret symmetric key, K_{A-B} or $K_{Alice-Bob}$, which is used for authentication. The other case is where Alice and Bob know each other's public keys as well as their own private keys. So, in each of these cases, we'll discuss how to establish a session key. Consider the first case where Alice and Bob share a secret

symmetric key, $K_{Alice-Bob}$ or K_{A-B}, which is used for authentication. Assume for concreteness that Alice and Bob use the protocol ap4.0, which is shown in this figure for one-way authentication.

Recall that in this protocol, Alice sends 'I am Alice,' then Bob responds with a nonce R, and Alice sends $K_{A-B}(R)$. We discussed many other protocols, so the cases where mutual authentication is done or timestamps are used can be handled similarly to our discussion for ap4.0, which we now discussed. The question is, after executing this protocol ap4.0, what can Alice and Bob use as a session key? So, let's consider several attempts. Suppose Alice and Bob use the value $K_{A-B}(R)$ as a session key after executing protocol ap4.0.

- After executing protocol ap4.0, suppose Alice and Bob use $K_{Alice-Bob}(R)$ as the session key

So, is this a secure protocol? Clearly, it is not secure because an intruder who eavesdrops on this conversation between Alice and Bob can find out $K_{A-B}(R)$, and hence the intruder can decrypt all the messages subsequently exchanged between Alice and Bob. So, K_{A-B} is sent on the channel; hence, it is not secure to use it as a session key. So, this attempt fails. We cannot use $K_{A-B}(R)$ as a session key.

Shared Symmetric Key (contd.)

- After executing protocol ap4.0, suppose Alice and Bob use $K_{Alice-Bob}(R+1)$ as the session key

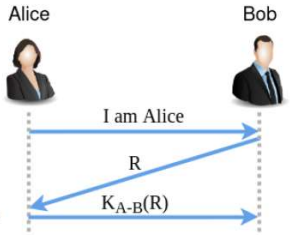
So, notice that in this context, the session key has to be something, which is known to Alice as well as Bob, or which can be computed by Alice and Bob. So, K_{A-B} has the advantage that it is known to Alice and Bob. But it has this drawback: it is sent over the channel, so it is known to an intruder who used it on the channel; so, it is not secure to use this as a session key. Consider another attempt: after executing protocol ap4.0, which is shown here, Alice and Bob use $K_{A-B}(R+1)$ as a session key. So, the nonce used during the authentication is R, but Alice and Bob use $K_{A-B}(R+1)$ as the session key.

So, this is something that can be computed by Alice and Bob because they know the secret K_{A-B}. So, each one can compute $K_{A-B}(R+1)$ independently. They will agree on the same value, $K_{A-B}(R+1)$. So, can they use this as the session key? So, is this a secure protocol?

It is not a secure protocol because suppose Trudy records the entire conversation that takes place using $K_{A-B}(R+1)$ as a session key. Later on, Trudy impersonates Bob's network address and waits for Alice to authenticate. When Alice attempts to authenticate herself, Trudy sends R+1 as the nonce to Alice, and the correct response from Alice is $K_{A-B}(R+1)$. So now, Trudy has obtained the value of $K_{A-B}(R+1)$. Trudy then uses this value to decrypt the entire conversation that took place using $K_{A-B}(R+1)$ as a session key.

So, Trudy is able to obtain the value $K_{A-B}(R+1)$ by sending R+1 as a nonce in an authentication exchange. Once Trudy obtains the value $K_{A-B}(R+1)$, she is able to use that value to decrypt the entire conversation that took place using $K_{A-B}(R+1)$ as a session key. So, from this example, we see that the session key should not be of the form $K_{A-B}(X)$, where X is some nonce that might be used in the authentication protocol. So, we want an alternative approach for selecting the session key. After executing protocol ap4.0, suppose Alice and Bob use $(K_{A-B} + 1)(R)$ as the session key.

- After executing protocol ap4.0, suppose Alice and Bob use $(K_{Alice-Bob}+1)(R)$ as the session key
- Is this a secure protocol?
  - ❑Yes
- More generally, Alice and Bob can use $g(K_{Alice-Bob})(R)$ as the session key, where $g(K_{Alice-Bob})$ is some modified version of $K_{Alice-Bob}$



So, what this means is: this is the value obtained by encrypting the nonce R using the secret key $(K_{A-B} + 1)$. So, this $(K_{A-B} + 1)$ is selected to be something other than the secret key K_{A-B}. So, is this a secure protocol? First, we note that Alice and Bob know K_{A-B}, so they can both calculate $(K_{A-B} + 1)(R)$ independently and reach the same value, $(K_{A-B} + 1)(R)$. Only Alice and Bob know K_{A-B}, so no one else can compute $(K_{A-B} + 1)(R)$. Also notice that this $(K_{A-B} + 1)(R)$ is not of the form of any message, which is sent over the channel during an authentication.

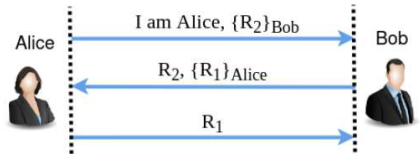
So, hence, the previous attack that we discussed cannot be used to extract the value of $(K_{A-B} + 1)(R)$. So, for these reasons, this is a secure protocol. More generally, Alice and Bob can use $g(K_{A-B})(R)$ as a session key, where $g(K_{A-B})$ is some modified version of K_{A-B}. For example, $g(K_{A-B})$ might be $(K_{A-B} + 10)(R)$ or it might be $(K_{A-B}^2)(R)$, and so on. So, it should

not be of the form $K_{A-B}(X)$, where X is some value. So, if it is of the form $K_{A-B}(X)$, then the protocol becomes vulnerable to the attack that we discussed in the previous slide.

So, this is one protocol which is secure. So, they can agree on a session key using this protocol. Now, we have discussed the first case, where Alice and Bob know a secret key, K_{A-B}, and we discussed how they can agree on the session key. Now, consider the other case. Suppose Alice and Bob authenticate each other using the protocol in this figure.

Alternatively, Alice sends 'I am Alice, R_2' to Bob; then Bob responds with R_2 signed with his key, that is signed with his private key and R_1; then Alice responds with R_1 signed with her key, that is signed with her private key. So, this is the case; these protocols can be used when Alice and Bob have a public key and a private key, and the public key of Alice is known to Bob, and the public key of Bob is known to Alice. So, either of these protocols can be used. Now, the question is, after executing one of these protocols, what can Alice and Bob use as the session key? So, suppose, to establish the session key, Alice selects a random number R, encrypts it using Bob's public key to get $\{R\}_{Bob}$, and sends it to Bob.

- Suppose to establish session key, Alice selects a random number, R, encrypts it using Bob's public key to get $\{R\}_{Bob}$ and sends it to Bob
- Alice and Bob use R as the session key
- Is this protocol secure?
  - No; an intruder, Trudy, on path between Alice and Bob can replace $\{R\}_{Bob}$ with $\{T\}_{Bob}$, where T is selected by Trudy, and send $\{T\}_{Bob}$ to Bob



Then Alice and Bob use R as the session key. So, someone who eavesdrops on the channel is not able to get the secret R because it is encrypted using Bob's public key. So, only Bob can decrypt it and obtain the value R. Hence, it leads us to think that this may be a secure protocol. But this protocol is not secure because we notice that an intruder, Trudy, on the path between Alice and Bob can replace $\{R\}_{Bob}$ with $\{T\}_{Bob}$, where T is selected by Trudy, and the intruder can send $\{T\}_{Bob}$ to Bob. So, notice that to compute $\{T\}_{Bob}$, the intruder only needs to know Bob's public key, which is well known.

So, the intruder can calculate $\{T\}_{Bob}$. Hence, this protocol can be attacked easily. An intruder can send $\{T\}_{Bob}$ to Bob, and Bob thinks that he's communicating with Alice,

whereas actually, the secret used in this communication is known to Trudy, not to Alice. So, the protocol is not secure for this reason. Now, the protocol on the previous slide failed because an intruder was able to replace the quantity $\{R\}_{Bob}$ with $\{T\}_{Bob}$.

So, the intruder was able to modify the quantity sent by Alice. So, we must add message integrity to this protocol. So, we must add a mechanism such that an intruder is not able to modify the message sent by Alice. Suppose to establish a session key, Alice selects a random number R and encrypts it using Bob's public key to get $\{R\}_{Bob}$, as in the previous protocol. But now Alice signs the result to get this, $\{R\}_{Bob}$ signed by Alice's private key, and sends the result to Bob.

Then Alice and Bob use R as the session key. So, this protocol is secure because an intruder Trudy on the path between Alice and Bob cannot forge Alice's signature. To create Alice's signature, one needs to know Alice's private key, which is not known to anyone but Alice. So, for this reason, this protocol is secure. So, when Alice and Bob have a public key and a private key, they can agree on a session key using this protocol.

Now, suppose Alice and Bob have established a session key using one of the protocols that we discussed. Then, they can use the session key for achieving confidentiality and message integrity of the data exchange that follows the authentication exchange. For example, a user might authenticate to an email server and subsequently they can secure the conversation that takes place using the session key they agreed upon during authentication. Generally, two session keys are established, one for confidentiality and one for message integrity. The reason is that if the same session key is used for both confidentiality and message integrity, then if an intruder who knows which algorithms are used for confidentiality and message integrity and the fact that the same session key is used for both algorithms, the intruder may be able to use this fact and the relation between the two algorithms to find some weakness.

So, hence, it's good security practice to use two different session keys; one for confidentiality and one for message integrity. Each packet is encrypted using the session key used for confidentiality and also a message authentication code is added to each packet for message integrity. Recall that if the message is m then the message authentication code is H(m,s). So, the message is m and the MAC is H(m,s), where s is the secret session key shared between the sender and receiver, which is used for message integrity. Now, consider the sequence of packets being sent from Alice to Bob.

So, every packet is protected with a MAC. So, if an intruder tries to create a new packet and send it to Bob, then Bob will reject the packet because the MAC check will fail. But an intruder on the path between Alice and Bob can still replay an old packet, that is, take an old packet and play it again, or delete a packet, or reorder packets. So, adding MACs individually to packets does not defend against these attacks: replaying, deletion, or reordering. That's because someone can take an old packet whose MAC is legitimate—it was created by Alice, and then replay it.

There is no way for Bob to check whether it's a new packet or a replayed packet. Similarly, someone can delete a packet, and Bob won't know, or someone can reorder packets. So, how do we defend against these attacks? So, one way is this: Alice includes a sequence number in each packet before computing the MAC. So, instead of finding (m, H(m,s)), Alice adds a sequence number, let's call it n, and computes H(m,n,s). So, this sequence number n might be 1 for the first packet, 2 for the second packet, 3 for the third packet, and so on.

So, Alice includes a sequence number in each packet before computing the MAC. So, if someone replaced the message, then they'll have to calculate the value H(m,n,s), where n is the new sequence number, but they won't be able to calculate this value because the secret s is unknown. So, the addition of a sequence number defends against these attacks: replay, deletion, and reordering. Another defense is that the MAC is computed over not just the current message but all messages from the beginning of the session until the current message. So, suppose the first message is m_1, the second message is m_2, and so on.

So the MAC that is computed for message m_n is this. So, m_n is the n^{th} message, and the hash values m_1 up to m_n, s, that is, (m_n, H(m_1,...,m_n), s). So, all the messages sent so far are concatenated, then we append the secret key s, and then compute the hash over that. So, this is another technique, which defends against these attacks: reordering, replay, and deletion. Because if someone replays an attack, this string will change. m_1 to m_n, this will change, or if someone deletes one message from these, then this will change, or if someone reorders packets, then this information, m_1 to m_n, will change.

So, this is another alternative approach to using a sequence number, which also defends against this attack. Now, consider the bidirectional exchange of sequences of packets being sent between Alice and Bob. An intruder on the path between Alice and Bob can record a packet going in one direction and replay it in the other direction. So, if sequence numbers are used, for example, an intruder can take a packet of sequence number n, which is being

sent from Alice to Bob, and then replay it from Bob to Alice with the same sequence number n. So, what are the defenses against this attack?

So, one defense is we use sequence numbers in different ranges in the two directions. For example, we use sequence numbers in the range 1 to 1000 from Alice to Bob, and in the range 1001 to 2000 in the direction Bob to Alice. So, one cannot take a packet going in one direction and replay it in the other direction because the sequence number would be invalid. Another defense is this: we include a direction flag in each packet. So, there is a direction flag which is 0 in one direction and 1 in the other direction.

For example, the flag is 0 in packets sent from Alice to Bob, and the flag is 1 in packets sent from Bob to Alice. So, if someone takes a packet going in one direction and tries to send it in the other direction, then the direction flag will indicate that this is not a valid packet for that particular direction. Another defense is during session key establishment, we establish four session keys, one each for confidentiality and message integrity in each direction. So, in this case, because the session keys are different in the two directions, we cannot take a packet going in one direction and play it back in the other direction because the session keys are different for the two directions. So, during session key establishment, we can establish four session keys.

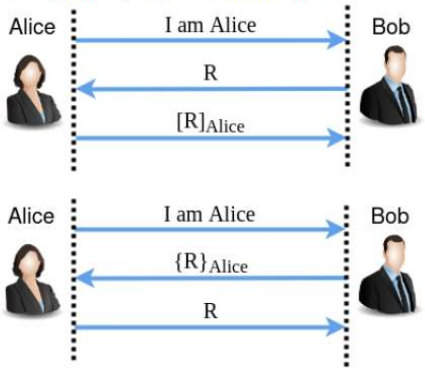
So, it is not difficult to establish four session keys during authentication. One can just agree on a particular session key and then break it into four parts, for example, or use some function to create four keys out of one shared key. So, this concludes our discussion of how communication can take place after authentication. Now, we discuss some other attacks related to authentication using public keys. So, the first attack is as follows.

Alice has a public key and private key pair, with the public key being known to Bob. Recall that Alice can authenticate herself to Bob using one of the protocols in this figure. In the first protocol, Alice sends ‘I am Alice’, then Bob responds with a nonce R. Alice signs the nonce and sends the response to Bob. In the second protocol, Alice sends ‘I am Alice’ to Bob. Bob sends $\{R\}_{Alice}$, which is R encrypted using Alice’s public key, and sends it to Alice.

Alice decrypts it using a private key and sends the nonce R to Bob. Now, suppose an intruder Trudy has a quantity X, on which she wants Alice’s signature. And suppose the first protocol is used for authentication. Then, Trudy can obtain Alice’s signature fraudulently as follows. She impersonates Bob’s network address and waits for Alice to log in.

Then, when Alice tries to log in, Trudy sends X as the nonce. X is the quantity on which Trudy wants Alice's signature. Then, the correct response is $[X]_{Alice}$. So, when Alice sends $[X]_{Alice}$ to Trudy, Trudy has obtained the signature of Alice on the document X. This protocol can be used for obtaining the signature of Alice on a quantity X on which Trudy wants Alice's signature. So, this is one attack, which is possible if this authentication protocol is used.

- Suppose an intruder, Trudy, has a quantity X, on which she wants Alice's signature
- Trudy can obtain it as follows:
  - She impersonates Bob's network address and waits for Alice to log in
  - Then sends X as nonce
  - Alice responds with $[X]_{Alice}$
  - (This assumes that protocol in first fig. is being used for auth.)



So, this assumes that the protocol in this first figure is being used for authentication. Another attack is as follows. Now, assume that this protocol in the second figure is being used for authentication. And the intruder, Trudy, has a quantity $\{X\}_{Alice}$, which is some quantity X encrypted using Alice's public key. And the intruder wants to find the quantity X.

So, Trudy doesn't have Alice's private key, so Trudy is not herself able to decrypt $\{X\}_{Alice}$. But Trudy can find X as follows. Again, Trudy impersonates Bob's network address and waits for Alice to log in. Then, Trudy sends $\{X\}_{Alice}$ as the challenge. That's done in this step.

Then, Alice decrypts the challenge $\{X\}_{Alice}$ and sends the decrypted version X. Now, Trudy has obtained the value X, which she wanted to obtain. So, this assumes that the protocol in this second figure is being used for authentication. So, how can we defend against these attacks? So, these attacks are possible because an intruder either wanted a signature of Alice or wanted a quantity decrypted by Alice and they used the authentication protocol for that purpose. Actually, this is a nonce; it is meant only for authentication, but someone was able to reuse this authentication protocol for fraudulently getting a signature or for decrypting some information.

So, we can defend against such attacks by ensuring that nonces and other quantities that may be signed or decrypted, for example, email messages, have different structures. By different structure, here's an example of such a different structure. Suppose Alice signs different types of information. She signs a nonce R in an authentication protocol like on the previous slide, as well as Alice signs an email message for achieving message integrity of that email message. So, we can concatenate a 'type field' to each quantity, which indicates what kind of quantity it is.

Is it a nonce, or is it an email message, or is it a file, and so on. So, the type field indicates what kind of quantity it is. So, we concatenate a type field to each quantity. So, that type field indicates what kind of quantity it is. This protocol defends against the attacks that we discussed because, let's go back to that slide.

So, in the first protocol, if Trudy wants signature of Alice, and if Trudy sends X to Alice, then by looking at the type field, Alice will know that it is not a nonce but it is some other kind of message like an email message, or a file. So, by looking at the type field, Alice will see that it is not a nonce and refuse to sign the message. Similarly, here when Trudy sends $\{X\}_{Alice}$, then Alice will know that this X is not a nonce, but it is some other kind of quantity, so Alice will not sign this. Hence, this defense of concatenating a type field works in this case. So, using this defense, we can defend against the attacks that we discussed.

So, in summary, we discussed how communication can take place after authentication, and we discussed different attacks related to public authentication, and we discussed the defense against such attacks. Thank you.