# Network Security
**Professor Gaurav S. Kasbekar**
**Department of Electrical Engineering**
**Indian Institute of Technology, Bombay**
**Week - 06**
**Lecture - 34**
## IPsec and Virtual Private Networks (VPNs) for Network-Layer Security: Part 3

Hello, in this lecture, we will continue our discussion of IPsec and virtual private networks for network layer security. Recall that if we want network layer security from a router R1 to a router R2, then first we establish two logical connections called security associations, one from R1 to R2 and one from R2 to R1. These are unidirectional logical connections. But the question is, how are these security associations created? So, they are created using a protocol known as the Internet Key Exchange Protocol (IKE).

We'll discuss that in this lecture. So, we discussed the establishment of security associations using the Internet Key Exchange Protocol. So, recall that to establish an IPsec security association from router R1 to router R2, they first need to authenticate each other; that is, router R1 needs to prove to router R2 that it is indeed the legitimate R1 and vice versa. They need to agree on the encryption and message integrity algorithms, which will be used. For example, DES or AES might be used for encryption, and SHA-1 might be used for the hash that is used to create the message authentication code for message integrity, and so on.

So, they agree on the encryption and message integrity algorithms, the keys, as well as the security parameter index. Recall that the SPI is used to identify an SA. So, these tasks are done using the Internet Key Exchange Protocol. So, it has two phases. We'll discuss the Internet Key Exchange Protocol now.

So, in phase one, the following actions are performed. The two sides use the Diffie-Hellman algorithm, which we discussed earlier, to create a bidirectional Internet Key Exchange security association between the routers. So, this is different from the IPsec security associations that we discussed earlier. Recall that IPsec security associations are unidirectional. In contrast, this IKE security association is bidirectional.

And it is different from the SAs that are created as part of IPsec. So, in phase one, first a bidirectional IKE security association is created between the routers. So, this IKE security association provides encryption and message integrity between the two routers. So, once the IKE security association is created, then a secure channel is now available, which provides encryption and message integrity between the routers. Then, over the secure channel, both sides reveal their identity to each other.

But since this information is encrypted, the identities are not revealed to a passive sniffer because the messages are sent over the secured IP security association channel. So, what is meant by this identity? One form of identity is the IP address. Here, we don't mean the IP address. Typically, the two entities that communicate using IPsec, they have another identity apart from IP address, such as an email address.

So, this other identity is protected via this encryption, that is, used as part of IKE security association. So, first, this secure channel is created, and then the identities are revealed. In phase two, the two sides create an IPsec security association in each direction by negotiating the IPsec encryption and message integrity algorithms to be employed and establishing the secret keys that will be used for encryption and message integrity for the two security associations. So, this phase two is run periodically. So, the IPsec security association keys are changed periodically because it's good security practice to periodically change the session keys.

The two sides can then use the IPsec security associations to securely send packets, as we discussed earlier. So, in particular, they use the ESP protocol to securely transfer packets over the security associations. So, in summary, in phase one, a secure IKE security association is created, and then the two sides reveal their identities to each other. And then, from time to time, the two sides create an IPsec security association in each direction by negotiating the algorithms to be used for encryption and message integrity and the keys that will be used. So, this phase two is executed periodically.

So, periodically, the IPsec security associations are recreated with new keys that are used for encryption and message integrity. So, recall that when the IKE protocol is used, the identities of the two parties are not revealed to a passive sniffer. So, one form of identification is the IP address. But recall that the IP addresses of the two parties are known to an intruder because they are included in the IP headers in unencrypted form. So, packets are sent over the security association after adding an IP header.

And for routing purposes, the source IP address and the destination IP address are included in plain text form in the IP header. So, the IP addresses of the two parties who are communicating using IPsec, they are known to an intruder because they are included in the IP headers in unencrypted form. But the two parties may, and usually do, use an alternative form of identification, such as an email address. Suppose routers R1 and R2 are communicating with each other; so, apart from the IP address, which is sent in clear form, the two parties also have an alternative identification, such as an email address, and IKE protects the confidentiality of such alternative forms of identification, such as an email address, using encryption. So, only after the IKE secure channel is created is this identity, that is, the email address, revealed.

But since it is encrypted, it is not known to a passive sniffer. Now, the question is: why are two phases used in the IKE protocol? Why not create the secure channel and the keys in one phase? So, the reason is that it is good cryptographic practice to periodically change the cryptographic keys used by the two communicating parties. So, the keys used for encryption and message integrity should be periodically changed.

Because if the same key is used for a long time, then the intruder may keep on sniffing the channel and collect a lot of ciphertext encrypted using those keys, and then it may become easy to break the cipher. So, for this reason, it is not a good practice to encrypt a large amount of information using the same keys. So, it’s better to periodically change the keys. Hence, multiple IPsec security association instances are generated over time, which use different cryptographic keys. So, now the reason for using two phases in the IKE protocol is the following.

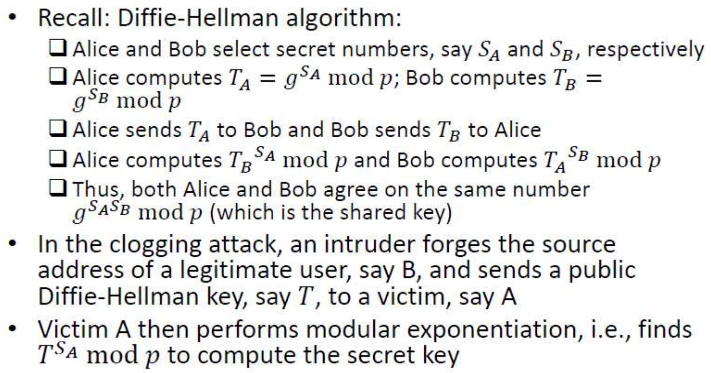
The reason is to save on computational cost. The reason that two phases save on computational cost is that phase 2 does not involve any public key cryptography. So, in phase 1, we use Diffie-Hellman, which is a public key cryptography technique, but in phase 2, there is no public key cryptography involved. So, phase 2 is much faster than phase 1. So, phase 1, which is executed only once, is used to generate long-term keys.

So, long-term keys are derived in phase 1. And later, phase 2 is executed from time to time to generate new IPsec security associations with new short-term keys. And these short-term keys are functions of the long-term keys generated in phase 1 and the nonces that are exchanged in phase 2. So, phase 1 is executed only once, right at the beginning. And in phase 1, long-term keys are generated, but these long-term keys are used very rarely.

Phase 2 is executed from time to time, and each time phase 2 is executed, new session keys are derived, which are used for encryption and message integrity over the security associations. So, these short-term keys are functions of the long-term keys computed in phase 1 and the nonces that are exchanged in phase 2. Since they are functions of nonces, they are different for different executions of phase 2. Hence, different keys are used each time phase 2 is executed. Now, this Internet Key Exchange Protocol is vulnerable to an attack known as the clogging attack.

We'll now discuss this attack. So, we call the Diffie-Hellman algorithm, which we discussed earlier. If Alice and Bob are executing the Diffie-Hellman algorithm, then they first select secret numbers, say, $S_A$ and $S_B$. So, Alice selects the secret number $S_A$, and Bob selects the secret number $S_B$, and these are known as the private keys of Alice and Bob. Then, Alice computes $T_A = g^{S_A}\ mod\ p$, which is the corresponding public key, and Bob computes $T_B = g^{S_B}\ mod\ p$, which is the corresponding public key of Bob.

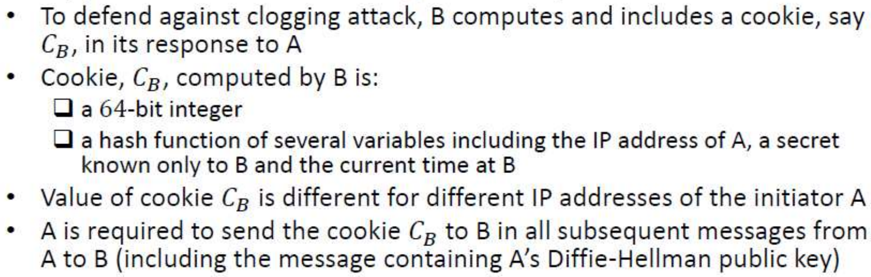
Then, Alice sends her public key $T_A$ to Bob, and Bob sends his public key $T_B$ to Alice. Then, Alice computes $T_B^{S_A}\ mod\ p$, and Bob computes $T_A^{S_B}\ mod\ p$. During our discussion of the Diffie-Hellman algorithm, we showed that these are equal and each is equal to this quantity $g^{S_A S_B}\ mod\ p$, and this is the shared key. So, via this scheme, Alice and Bob agree on the same number, $g^{S_A S_B}\ mod\ p$, which is the shared key. Now, what is the clogging attack? So, in the clogging attack, an intruder forges the source IP address of a legitimate user, let's call that user B, and sends a public Diffie-Hellman key, let's call that T, to some victim. We denote the victim by A.



- Recall: Diffie-Hellman algorithm:
  - ❑ Alice and Bob select secret numbers, say $S_A$ and $S_B$, respectively
  - ❑ Alice computes $T_A = g^{S_A} \bmod p$; Bob computes $T_B = g^{S_B} \bmod p$
  - ❑ Alice sends $T_A$ to Bob and Bob sends $T_B$ to Alice
  - ❑ Alice computes ${T_B}^{S_A} \bmod p$ and Bob computes ${T_A}^{S_B} \bmod p$
  - ❑ Thus, both Alice and Bob agree on the same number $g^{S_A S_B} \bmod p$ (which is the shared key)
- In the clogging attack, an intruder forges the source address of a legitimate user, say B, and sends a public Diffie-Hellman key, say $T$, to a victim, say A
- Victim A then performs modular exponentiation, i.e., finds $T^{S_A} \bmod p$ to compute the secret key

So, the intruder forges some IP address and sends a public key, say T, to a victim, say A. Then, the victim A performs modular exponentiation, that is, finds T to the power SA mod p to compute the secret key, as in the Diffie-Hellman algorithm. But this is a computationally expensive operation. So, this modular exponentiation is computationally expensive. Hence, by repeatedly sending messages of this type, the intruder can clog the victim with useless work of computing this key. So, repeated messages of this type can clog the victim's system with useless work, so that's the clogging attack. So, an intruder just keeps on repeatedly sending such messages and forces the victim to compute the secret key, which involves a lot of operations.

So, to defend against this attack, a mechanism called cookies is used in the IKE protocol. So, we'll now discuss the use of cookies in the IKE protocol to defend against the clogging attack. And then we'll discuss the operation of the IKE protocol, which includes cookies. Suppose A sends the first message of the IKE protocol to B. To defend against the clogging attack, B computes and includes a cookie, let's call it $C_B$, in its response to A. The cookie $C_B$ that is computed by B is a function of the following quantities.



- To defend against clogging attack, B computes and includes a cookie, say $C_B$, in its response to A
- Cookie, $C_B$, computed by B is:
  - ❑ a 64-bit integer
  - ❑ a hash function of several variables including the IP address of A, a secret known only to B and the current time at B
- Value of cookie $C_B$ is different for different IP addresses of the initiator A
- A is required to send the cookie $C_B$ to B in all subsequent messages from A to B (including the message containing A's Diffie-Hellman public key)

It's a 64-bit integer, and it is a hash of several variables, including the IP address of A, a secret known only to B, and the current time at B. So, the cookie is a function of these. So, notably, it's a function of the IP address of A, which is the node that sent the first message of the IKE protocol to B. And it's also a function of a secret known only to B. Hence, no one other than B can create the correct value of the cookie. The value of the cookie $C_B$ is different for different IP addresses of the initiator A. So, if we replace the IP address of A with some other IP address, then the value of the cookie will come out to be different with a high probability.

Now, the cookie is used in the following way. A is required to send the cookie $C_B$ to B in all subsequent messages from A to B, including the message containing A's Diffie-Hellman public key. So, the message containing A's Diffie-Hellman public key is received by B, and if it is okay, then B performs the modular exponentiation operation. But before performing the modular exponentiation operation, B verifies whether the cookie value is correct. On receipt of a message from A, B checks whether the cookie corresponds to A's IP address.

If the check fails, then B aborts the session establishment. Hence, B avoids performing the expensive modular exponentiation step of Diffie-Hellman. So, in this way, we can prevent an intruder from forcing the victim B to perform a modular exponentiation, which is expensive. Now, can an attacker who is not the legitimate A easily obtain the cookie $C_B$? An attacker can only obtain the cookie $C_B$ if it sniffs the channel between B and A, which is relatively difficult to do.

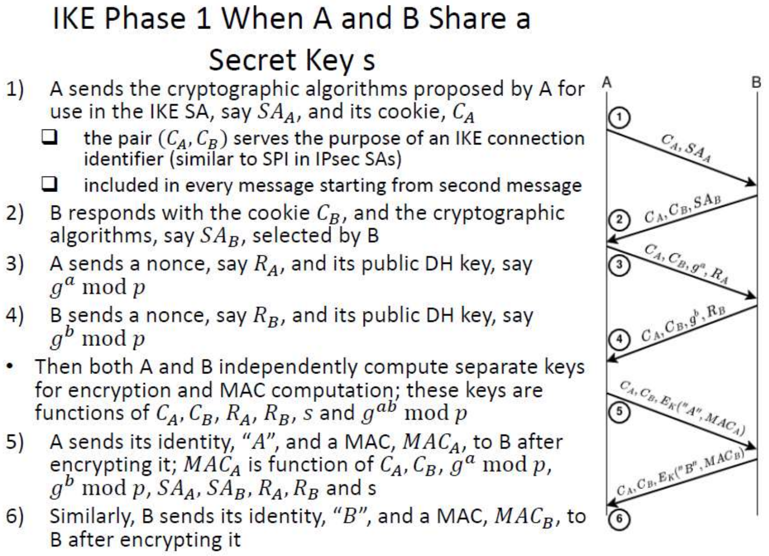
So, it's easy to spoof another IP address and send a message pretending to be from some other IP address, but it's relatively difficult to sniff the channel between B and A. The cookie is sent from B to A, so the only way the attacker can obtain the cookie is by sniffing the channel between B and A, which is difficult. So, the attacker might be somewhere on the internet. The attacker may not necessarily control a compromised router between A and B. Hence, it becomes difficult for the attacker to obtain the cookie $C_B$. Does the use of a cookie defend against a clogging attack? So, we now discuss the IKE protocol itself, which incorporates the use of cookies.

So, in phase 1, the following cases may arise. In the first case, A and B share a secret key, let's call it s, which is possibly derived from some password, or the other possible case is that A and B have a public-private key pair and certificates. So, we'll study this case 1 in detail, and case 2 is similar, so we won't go into the details of case 2. Assume that A initiates the IKE protocol and B is the responder. So, this is the execution of IKE phase 1, when A and B share a secret key s. This picture shows the execution.

There are six steps involved in this protocol. A is the initiator. In the first step, A sends the cryptographic algorithms proposed by A for use in the IKE SA. Let's call that information $SA_A$. A also sends its cookie, $C_A$, which is computed using the rules that we discussed on the previous slide.

So, $SA_A$ contains the list of cryptographic algorithms proposed by A for use in this security association. Now, recall that a cookie is also going to be sent by the responder B, that cookie is $C_B$. So, this pair ($C_A$, $C_B$) serves the purpose of an IKE connection identifier. It is similar to the security parameter index in IPsec security associations. So, this pair ($C_A$, $C_B$) is included in every subsequent message.

So, it's included in every message starting from the second message. So, we can see here that $C_A$, $C_B$ is there in each of these five messages. Now, in the second step, B responds with the cookie $C_B$ and the cryptographic algorithms, say $SA_B$, selected by B. So, this list of cryptographic algorithms proposed by A contains several possible encryption algorithms, several message identity algorithms, and so on. So, B selects one encryption algorithm, one message integrity algorithm, and so on from this list and includes that in the values $SA_B$ and sends it in its response to A. So, that's the second message, that is this one. Notice that this negotiation of the algorithms that will be used for security is quite similar to that in SSL.



IKE Phase 1 When A and B Share a Secret Key s

1) A sends the cryptographic algorithms proposed by A for use in the IKE SA, say $SA_A$, and its cookie, $C_A$
   - ❑ the pair $(C_A, C_B)$ serves the purpose of an IKE connection identifier (similar to SPI in IPsec SAs)
   - ❑ included in every message starting from second message
2) B responds with the cookie $C_B$, and the cryptographic algorithms, say $SA_B$, selected by B
3) A sends a nonce, say $R_A$, and its public DH key, say $g^a \bmod p$
4) B sends a nonce, say $R_B$, and its public DH key, say $g^b \bmod p$

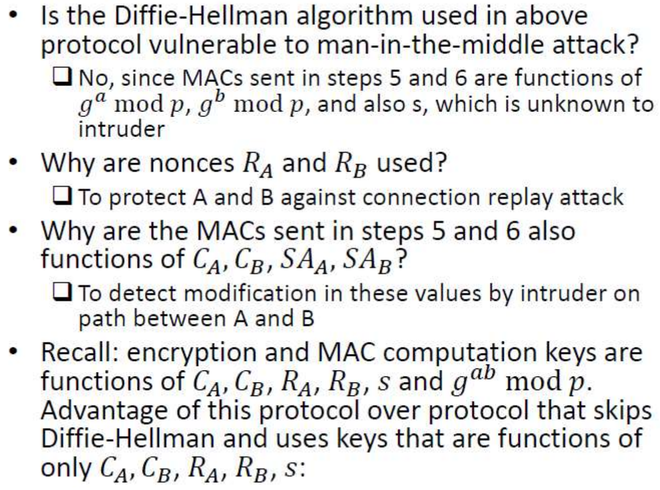
- Then both A and B independently compute separate keys for encryption and MAC computation; these keys are functions of $C_A, C_B, R_A, R_B, s$ and $g^{ab} \bmod p$

5) A sends its identity, "$A$", and a MAC, $MAC_A$, to B after encrypting it; $MAC_A$ is function of $C_A, C_B, g^a \bmod p$, $g^b \bmod p, SA_A, SA_B, R_A, R_B$ and s
6) Similarly, B sends its identity, "$B$", and a MAC, $MAC_B$, to B after encrypting it

In the third step, A sends a nonce, let's call it $R_A$, and its public Diffie-Hellman key, say $g^a \ mod \ p$. So, A is the private Diffie-Hellman key. Similarly, in step 4, B sends a nonce, let's call it $R_B$, and its public Diffie-Hellman key, we call it $g^b \ mod \ p$, where b is B's private Diffie-Hellman key. So, at this point, A and B have exchanged the necessary material to derive their keys. A and B independently compute separate keys for encryption and MAC computation, and these keys are functions of all these values: $C_A$, $C_B$, $R_A$, $R_B$, s, and $g^{ab} \ mod \ p$. So, notice that after exchanging the Diffie-Hellman public keys, they can agree upon a shared secret key, $g^{ab} \ mod \ p$. So, each one of A and B can find out this value, $g^{ab} \ mod \ p$. And they independently compute separate keys for encryption and MAC computation, which are functions of all these quantities.

Now, at this point, a secure channel has been created between A and B. A sends its identity, let's call it "A", and a message authentication code, say $MAC_A$, to B after encrypting it. And $MAC_A$ is a function of $C_A$, $C_B$, $g^a \ mod \ p$, $g^b \ mod \ p$, $SA_A$, $SA_B$, $R_A$, $R_B$, and s. So, this MAC is used to verify whether there has been any tampering of all these messages that were sent before message 5. Similarly, in step 6, B sends its identity, let's call it "B", and a message authentication code, $MAC_B$, to A after encrypting it. So, that is shown in this message 6. So, this is B's identity and a MAC is sent for verification.

Again, this $MAC_B$ checks if there has been any tampering of the messages that are sent in messages 1, 2, and 5. So, let's now analyze this protocol. So, the first question is, is this Diffie-Hellman algorithm used in the protocol vulnerable to the man-in-the-middle attack? So, we discussed the man-in-the-middle attack on Diffie-Hellman, where some intruder can modify the public key that is sent by A and then send the modified public key to B and vice versa. So, the intruder can take a modified public key from B and send the modified public key to A. So, is the Diffie-Hellman algorithm vulnerable to the man-in-the-middle attack?



- Is the Diffie-Hellman algorithm used in above protocol vulnerable to man-in-the-middle attack?
  - ❑ No, since MACs sent in steps 5 and 6 are functions of $g^a \bmod p$, $g^b \bmod p$, and also s, which is unknown to intruder
- Why are nonces $R_A$ and $R_B$ used?
  - ❑ To protect A and B against connection replay attack
- Why are the MACs sent in steps 5 and 6 also functions of $C_A, C_B, SA_A, SA_B$?
  - ❑ To detect modification in these values by intruder on path between A and B
- Recall: encryption and MAC computation keys are functions of $C_A, C_B, R_A, R_B, s$ and $g^{ab} \bmod p$. Advantage of this protocol over protocol that skips Diffie-Hellman and uses keys that are functions of only $C_A, C_B, R_A, R_B, s$:

It is not because, consider this public key, $g^a\ mod\ p$, if this is modified by an intruder to some other value, then that modification will be detected in the message authentication codes that are sent in steps 5 and 6, which are functions of all these quantities that were sent in messages 1, 2, 4, including $g^a\ mod\ p$, $g^b\ mod\ p$, and also s, which is unknown to an intruder. So, an intruder cannot generate the MACs corresponding to the modified messages because s, which is the long-term secret key, is unknown to the intruder. So, the correct MACs can only be generated by A and B. And these MACs are functions of, among other quantities, $g^a\ mod\ p$ and $g^b\ mod\ p$. So, any modification in them can be detected. So, hence, the Diffie-Hellman algorithm is not vulnerable to the man-in-the-middle attack. Now, another question is, why are nonces $R_A$ and $R_B$ used?

So, $R_A$ defends A against connection replay attacks, and $R_B$ defends B against connection replay attacks. So, the use of nonces is similar to the use in SSL, in the SSL handshake, which we discussed in an earlier lecture. So, for the same reason, it is used, namely to protect A and B against the connection replay attack. Now, why are the MACs sent in steps 5 and 6 also functions of $C_A$, $C_B$, $SA_A$, and $SA_B$? So, the MACs are functions of these quantities to detect any modification in them by an intruder on the path between A and B. For example, an intruder might remove the strong algorithms in the list $SA_A$ and force B to choose some weak algorithms.

So, such modification can be detected by having the MACs as functions of $C_A$, $C_B$, $SA_A$, and $SA_B$. So, these cannot be modified by an intruder. So, the receiver will detect the modification. Now, recall that the encryption and MAC computation keys are functions of $C_A$, $C_B$, $R_A$, $R_B$, s, and $g^{ab}\ mod\ p$. So, what if we use an alternative simpler protocol which skips the Diffie-Hellman exchange and uses keys that are functions of only $C_A$, $C_B$, $R_A$, $R_B$, and the long-term key s? So, the advantage of this protocol is that this protocol provides forward secrecy, whereas the protocol which skips Diffie-Hellman does not provide forward secrecy.

So, this is very similar to our discussion of ephemeral Diffie-Hellman in the context of the SSL handshake. So, there we discussed that the ephemeral Diffie-Hellman algorithm provides forward secrecy. In this case, if the long-term key s is compromised in the future sometime, then in the absence of Diffie-Hellman, where the keys are only functions of $C_A$, $C_B$, $R_A$, $R_B$, and s, the intruder can calculate the keys used for encryption and message integrity. And hence, all the subsequent data exchanges that happen can be decrypted by the intruder. Since the Diffie-Hellman algorithm is used, even if an intruder later on somehow gets the secret key s, even then, just by recording the messages exchanged over the channel during these steps, the intruder cannot calculate the value $g^{ab}\ mod\ p$ because a property of Diffie-Hellman is that just by using the values exchanged over the channel, one cannot calculate the shared secret key, $g^{ab}\ mod\ p$. So, it is computationally infeasible to calculate $g^{ab}\ mod\ p$. So hence, the use of Diffie-Hellman results in forward secrecy. That is, sometime in the future, if an intruder gets the secret key s and if the intruder has recorded all these messages exchanged in these six steps, even then, the intruder is not able to calculate the secret key used for encryption and message integrity and hence is not able to decrypt the following messages after this handshake. So that's the reason for using Diffie-Hellman. Now, there's a major drawback with this option in which A and B share a secret key s. The drawback is as follows. When B receives this message 5, it must first decrypt the message to check the identity of A because the identity of A is encrypted.

But for decrypting the message, B needs to know the secret s that it shares with A. Now, to know s, B clearly needs to know the identity of A, that is, this “A”. So, A is itself encrypted, so how does B know the identity of A? So, how does B know which is the secret key s to be used for decrypting this message? So, one option is to use the source IP address to identify the sender.

So, by looking at the source IP address, B finds out the identity of A and then uses the corresponding key, that is s, which is shared between A and B. But a shortcoming of this is that if the source IP address reveals the sender’s identity, then there was no point in protecting the sender’s identity “A” using encryption. So, that’s a shortcoming of this approach. So, a better option is that B keeps track of all the entities that may communicate from a given source IP address. From the point of view of B, there may be multiple entities which communicate from the same source IP address, and B keeps track of all of them. Then, when message 5 arrives from a particular source IP address, B attempts to decrypt the message using the secrets that it shares with each of the entities at that IP address.

For example, if there are five entities that may possibly communicate from A’s source IP address, then B sequentially tries out the secrets that it shares with each of these five entities. Until successful decryption happens, which occurs when the key used for decryption matches that shared by the entity whose ID appears in this message 5, that is at this place. So, this is where the ID appears. If this ID is that of the entity whose key was used for decryption, in that case, the decryption is successful. So, this is the resolution to this shortcoming, that is, there are multiple entities which may communicate from a particular source IP address, and the recipient tries out the secret keys that it shares with all of those entities until successful decryption happens.

So, we have discussed IKE phase 1. Now, we discuss IKE phase 2. So, recall that after IKE phase 1, a channel that provides encryption and message integrity has been created between A and B. This channel is the IKE security association. In phase 2, the two sides create an IPsec security association in each direction by negotiating the IPsec encryption and message integrity algorithms to be employed by the IPsec security associations and establishing the encryption and message integrity session keys for the two IPsec security associations. So, this phase 2 is periodically run to negotiate which algorithms will be used for encryption and message integrity and for deciding on what keys will be used.

The two sides can then use the IPsec security associations to securely send packets using the ESP protocol, as discussed earlier. So, this IKE phase 2 is executed from time to time to create a new IPsec SAs, since it's considered a good cryptographic practice to change the keys from time to time. So, in summary, we discussed IPsec and the Internet Key Exchange protocol. So, in the Internet Key Exchange protocol, the secure channel is created, and then, from time to time, new security associations are created over which the two entities can securely communicate using the ESP protocol. So, this concludes our discussion of IPsec and virtual private networks for network layer security.

Thank you.