Network Security

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

Week - 03

Lecture - 16

Message Integrity, Cryptographic Hash Functions and Digital Signatures: Part 1

Hello, recall that in the last few lectures, we discussed the principles of cryptography. In this lecture and the next few lectures, we will discuss message integrity, cryptographic hash functions, and digital signatures. So, before we describe the general message integrity problem, let's consider an example. Consider a network as illustrated in this figure here. And this network uses Dijkstra's algorithm to find routes between every pair of routers.

We discussed Dijkstra's algorithm briefly in the review of basic communication networks part. These blue icons are routers, and they are connected together by communication links. Before the operation of Dijkstra's algorithm, each router broadcasts a list of all its neighbors and the cost of the corresponding links to all the other routers. For example, this router U creates a list that says its neighbors are V, X, and W, and the cost of the corresponding links are 2, 1, and 5. Each router creates such a list and sends this list to all the other routers in the network.

So, each router broadcasts a list of all its neighbors and the cost of the corresponding links to all the other routers in the network. The message containing this list is called a link state message. Once every router has sent such a list to all the other routers in the network, each router then knows the entire network topology. That is, what are the different nodes, how they are connected to each other, and what are the costs of the corresponding links? So, each router then knows the entire network topology.

It uses this information to compute shortest paths using Dijkstra's algorithm. A key kind of message that is required for the execution of this algorithm is a link state message. It's a message sent by one router to the other routers containing a list of the neighbors of the router and the costs of the corresponding links. Now, is it possible for an intruder to attack such a routing algorithm? One way an intruder can attack such a routing algorithm is an intruder can send a bogus link state message to a router B with source address that of router

A. So, we discussed that it is possible in the internet to send a message to another router or another host with source IP address that of some other router.

So, it's easy to create a message with a false source IP address. So, using this fact, an intruder can send a bogus link state message to a router B with source IP address that of router A. The intruder claims to be router A and sends a bogus link state message to a router B, and it can cause router B to make unnecessary modifications in its routing table. So, for example, the intruder can modify the routing table at B in such a way that messages that reach B are forwarded to the intruder, and then the intruder might drop such messages. So, the intruder can make modifications in the routing table of router B, which are unnecessary modifications, by sending such a bogus link state message. Another way this routing algorithm can be attacked is an intruder might modify a link state message from A before forwarding it to B.

Suppose A sends a link state message towards B, and this message is intercepted by an intruder who modifies the message before forwarding it to B. So, such attacks are possible. An intruder can send a bogus link state message, or an intruder can modify a message before forwarding it to the destination. So, when router B receives a link state message with the source IP address of router A, B needs to verify that it was indeed router A who created the message and not someone else claiming to be A. And also, B needs to verify that the message was not modified while being forwarded from A to B. In this example, there must be some mechanism for B to be able to verify that the source of the message is indeed legitimate and the message was not modified while being forwarded from the legitimate source to B. So, in this example, we see the role of message integrity. The general message integrity problem is the following.

Some user, Bob, receives a message with the source address of Alice. The message might be encrypted or in plain text. Some messages might require encryption; others may not be confidential. They may be sent in plain text. In either case, Bob needs to verify that Alice actually created the message and not someone else pretending to be Alice.

Bob also needs to verify that even if Alice actually created the message, the message was not modified while being forwarded from Alice to Bob. So, Bob should accept the message only if it was actually sent by the claimed sender Alice and it was not modified during transit from Alice to Bob. So, this is a message integrity problem to add mechanisms to a message such that the receiver can verify that the sender is legitimate and the message was not modified during transit from the sender to the receiver. So, let's discuss several

attempts to achieve message integrity. Let m denote the message to be sent from Alice to Bob.

Suppose Alice performs the following actions. She computes the checksum of the message m. Let's call it c(m). For example, it might be an odd parity checksum, Hamming code, and so on. Alice sends the message m along with the checksum c(m) to Bob. Bob considers the m part of the received message, finds out its checksum, and checks whether the checksum of m is indeed c(m). If the checksum of m is c(m), Bob accepts the message.

The reason for adding this checksum is that if an intruder replaces this message m with some other message m', then c(m') may not be equal to c(m), and check performed at Bob will fail, so Bob will reject the message if an intruder replaces m with m'. Now, in this protocol, the checksum function is well known; that is, everyone knows which checksum function is used in this protocol. So, in this protocol, as well as in subsequent protocols, the checksum function is known to everyone. Only any secret key is confidential. So, in this protocol, there is no secret key.

So, does this protocol achieve message integrity? It does not achieve message integrity because an intruder, say Trudy, can create a bogus message, let's call it m', and since the checksum function is known to everyone, Trudy can find out the checksum of m' and send (m', c(m')) to Bob with the source address that of Alice. So, Trudy can spoof the source address and make the source address that of Alice and send this message (m', c(m')) to Bob. So, when Bob checks whether the checksum of m' is equal to c(m'), that check passes at Bob. So, Bob accepts the message.

So, hence, this attempt does not achieve message integrity. An intruder is able to modify the transmitted message or send a bogus message to the receiver. So, we need another way to achieve message integrity. Here's another possible attempt. Alice performs the following actions.

She computes a checksum of m, say c(m), and to hide the checksum, she encrypts it to get $K_A(c(m))$, and then sends $(m, K_A(c(m)))$ to Bob, where K_A is the secret key used to encrypt the checksum. So, $K_A(c(m))$ denotes the ciphertext obtained by encrypting c(m) using the secret key K_A. So, Alice sends $(m, K_A(c(m)))$ to Bob, the message along with the encrypted checksum. Then Bob first decrypts the checksum to get c(m) and then checks whether it equals the checksum of m. So, Bob computes the checksum of m and verifies whether that equals the decrypted checksum obtained by decrypting this part, $K_A(c(m))$. Again, the checksum function is known to everyone. The keys used in this encryption and decryption

protocol. The keys used for encryption and decryption are secret. So, does this protocol achieve message integrity?

- Alice performs the following actions:
  - computes checksum of m, say $c(m)$
  - encrypts it to get $K_A(c(m))$
  - sends $(m, K_A(c(m)))$ to Bob
- Bob finds $K_B(K_A(c(m))) = c(m)$ and checks whether it equals checksum of m

Now in this case, an intruder who intercepts this message is not able to directly find out the checksum c(m) because this part is encrypted. So, it's K_A(c(m)), and the intruder does not know the secret key to decipher this part. But notice that this message m is sent in clear form, in plaintext form. So, the intruder can find out the checksum of m because the checksum function is well known. So, the intruder can take this first part of the message m and find its checksum.

And intruder Trudy can then create a bogus message m' such that c(m') = c(m). Then the intruder can send the bogus message m' along with the original encrypted checksum, that is K_A(c(m)), to Bob with the source address that of Alice. Now, this message m' is chosen such that c(m') = c(m). When the receiver Bob finds out the checksum, the value obtained after decrypting this part of the message. So, c(m) = c(m'), and hence, the verification at Bob succeeds. So, Bob accepts the message. So, in this way, the intruder is able to create a bogus message and send it to Bob.

So, this attempt also fails because it is easy to find another message m' which has the same checksum as a particular message m. So, because of this property of the checksum, this attempt fails. It's easy to find, in general, another message which has the same checksum as a given message. So, let's try to fix this attempt. In attempt 3, Alice performs the following actions. She computes the checksum of m, say c(m), and then concatenates m and c(m) to get (m, c(m)) and then encrypts the entire concatenated message to get its encrypted version K_A(m, c(m)), and sends it to Bob.

Bob then decrypts the entire message to get back m and c(m) and checks whether the checksum of m equals c(m). Does this achieve message integrity? An intruder who intercepts this message obtains this value, K_A(m, c(m)). Since the intruder does not have the secret key for decryption, the intruder is not able to get either m or c(m). So, the intruder

is not able to decipher this message to get m or c(m). And the intruder is also not able to create another message (m', c(m')). Because if the intruder creates another message, a bogus message m' and finds its checksum c(m'), then the intruder will not be able to encrypt it because the intruder does not know the key required to encrypt a message and send it to Bob. So, the intruder is not able to create a bogus message and find its checksum and then encrypt the entire (m', c(m')) because the intruder does not have the secret key to encrypt and send a message to Bob.

So, hence, this protocol achieves message integrity. But this has a shortcoming that it requires a sender to encrypt the entire message m, which is time-consuming. In many cases, m may not be confidential. For example, a link state message sent in a routing protocol. So, the message may not be confidential.

Just for the sake of message integrity, we still need to encrypt the message. So, encryption adds overhead. So, it is time-consuming to encrypt the message. So, that's one shortcoming of this approach. Even when the message is not confidential, we need to encrypt it, which takes time.

Can we have an approach in which the message does not need to be encrypted? Another shortcoming is that in some cases encryption of the message may not be allowed. For example, security agencies may want to monitor all communications to see whether any terrorists are sending messages or any messages related to crime are being sent, and so on. So, security agencies may want to monitor all communications, so they may not permit the encryption of the messages. They may permit the encryption of the checksum but not of the message.

So, in such cases, this protocol cannot be used because the message is encrypted in this case. So, this attempt achieves message integrity, but it has some shortcomings, which we discussed. We want to send the message m in plaintext form and still achieve message integrity. So, the basic idea is this: recall attempt 2. So, attempt 2 fails because the intruder could create a bogus message m' such that c(m') = c(m). Checksum has this property that, in general, it's easy to find another message m' which has the same checksum as a given message m. Now, suppose we replace this checksum function with a function H(.), and H(.) has this property that given a message m, it is computationally infeasible to find another message m' such that H(m') = H(m).

In this case, attempt 2 would work because the attack we discussed on attempt 2 does not succeed because the intruder is not able to find another message m' which has the same H

value as the original message m. Such a function H(.) is called a cryptographic hash function. We now define a cryptographic hash function. A cryptographic hash function is a function H(.) which takes an input and finds an output such that for an input message m of arbitrary length, the output H(m) is a fixed-length string. So, the length of the output depends on which hash function we use. Some hash functions have an output length of 128 bits, others have 256 bits, some have 512 bits, and so on and so forth.

So, the output H(m) is a fixed length string for an input m of arbitrary length. This illustrates the computation of a hash function. The input is a message m, which might be a long message. It might be of arbitrary length. And we have a many-to-one hash function.

And it produces an output of fixed length. Another property of a cryptographic hash function is that given a message m, its hash value can be computed fast. Much faster than encrypting the message m. A crucial property of a cryptographic hash function is this. It is computationally infeasible to find a message m that is a pre-specified hash h. That is, a message m such that H(m) = h. So, for this reason, a cryptographic hash function is called a one-way function. Given an input, it's computationally easy to find out its hash value.

So, given a message m, it's straightforward to find its hash value H(m). But it's difficult to go in the other direction. That is, given a target hash value h, it's computationally infeasible to find the message m, which has the hash value h. Another property of a cryptographic hash function is this. It is computationally infeasible to find two messages m and m' such that H(m') = H(m). So, this is called a collision. Two messages m and m' such that they have the same hash value.

It's computationally infeasible to find two messages, m and m', which have the same hash value. That's another key property of a cryptographic hash function. Another key property related to the previous properties is the following. Given message m, it is computationally infeasible to find a different message m' which has the same hash value as the original message m. So, given m, it's computationally infeasible to find a different message m' such that H(m') = H(m). So, because of this property, if we replace a checksum function with a hash function in attempt 2, then it successfully achieves message integrity. Some popular cryptographic hash functions are SHA-1, SHA-2, and SHA-3.

These are abbreviations for Secure Hash Algorithm 1, 2, and 3. And SHA-1 collision was found in 2017. So, you can read the details at this link. That is, two different messages were found such that H(m) = H(m') for the two messages. Where H(.) is the SHA-1 hash function.

So, since a collision is found, it's considered not secure. Another popular cryptographic hash function is MD5, which stands for Message Digest 5. Several collisions were found between 2004 and 2007. So, for this reason, MD5 is considered insecure. Here's an example.

So, this is an example of a candidate cryptographic hash function. Suppose Bob owes Alice $100.99 and sends the following message to her. I owe you $100.99, Bob. So, this is a message saying that Bob owes Alice this amount. So, for records, Bob wants to send this message to Alice.

So, consider the following candidate cryptographic hash function. The ASCII representation of each character in this message is found. And then groups of 4 bytes each are added to get the checksum. This upper row shows the original message: "IOU100.99BOB". So, it is divided into groups of 4 bytes each.

And each byte is represented using its ASCII representation. For example, the ASCII representation of I is 49, then that of O is 4F, and so on. So, we find the ASCII representation of each byte. So, these ASCII representations are shown here, and then groups of 4 bytes each are added to get the checksum. So, these are added: 31, 39, and 42.

These are added, and if you add them in hexadecimal, then we get AC. Similarly, 55, 2E, and 4F are added to get the sum D2 in hex, and so on and so forth. So, this sum is the candidate hash value. So, we'll see that it's a valid checksum but it's not a valid cryptographic hash function. So, these groups of 4 bytes each are added to get the checksum.

Now consider, can we find a fraudulent message with the same checksum? So, that's straightforward because we can consider this; we can just flip two characters. So, this 1 and this 9 can be reversed to get 9 and 1. So, we create this fraudulent message: "IOU900.19BOB". So, it's straightforward to verify that this has the same checksum, B2 C1 D2 AC.

So, this message also has the same checksum as this message. Hence, this checksum function would make a poor cryptographic hash function. So, this is a valid checksum, but it's a poor cryptographic hash function. It's straightforward to find another message which has the same output as a given message. So, this would make a poor cryptographic hash function.

In practice, we use much more complicated functions. Later, we'll see an example of a practical cryptographic hash function. Now, consider the modified version of attempt 2. Alice performs the following actions. She computes the hash of m, say H(m), then encrypts it to get $K_A(H(m))$, and sends $(m, K_A(H(m)))$ to Bob.

In this protocol, note that the hash function is well known; that is, everyone knows which hash function is used for computing H(m). But the secret key used for encryption is not known to the public. So, the key used for encryption is a secret, assuming that symmetric key cryptography is used. If public key cryptography is used, then the public key is known to everyone, but the private key is a secret known only to the receiver. Bob decrypts this part of the message to get $K_B(K_A(H(m))) = H(m)$, and checks whether it equals the hash of m. So, this achieves message integrity.

- Alice performs the following actions:
  - computes hash of m, say $H(m)$
  - encrypts it to get $K_A(H(m))$
  - sends $(m, K_A(H(m)))$ to Bob
- Bob finds $K_B(K_A(H(m))) = H(m)$ and checks whether it equals hash of m

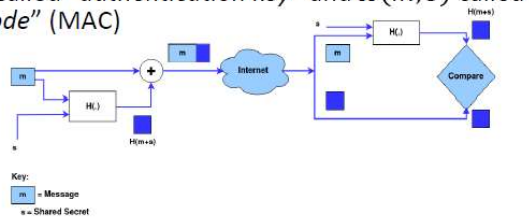
For example, if an intruder tries to replace this message m with another message (m', H(m')) will not match the decrypted version of this part of the message. So, this check will fail, and Bob will detect that the message has been modified. So, the property of the cryptographic hash function that we discussed above is crucial. It is computationally infeasible to find another message m', which has the same hash as the message m. So, it's computationally infeasible to find m' such that H(m) = H(m'). So, hence, an intruder cannot just replace this message m with another message m' because then this check performed by Bob will fail.

So, this protocol achieves message integrity, and it achieves message integrity because of the crucial property of a hash function that it's computationally infeasible to find another message m' which has the same hash value as a given message m. Now, the modified version of attempt 2, which we discussed on the previous slide, achieves message integrity, but that requires encryption, which is time-consuming. So, in this protocol, notice that Alice needs to encrypt the hash value of the message, m, which is time-consuming. Assume that Alice and Bob have a shared secret bit string, s. It can be possibly shared in advance

using public key encryption. Now, we want to achieve message integrity without using any encryption. That is, we don't want to encrypt the message, but we also don't want to encrypt any hash value.

So, consider such a protocol. Alice performs the following actions. She concatenates m and s to get (m, s) and then computes the hash value of the result. That is H(m, s). And then sends the original message m along with H(m,s) to Bob. So, Bob takes this m and concatenates it with s because Alice and Bob know this secret s. So, Alice concatenates m with s and then finds its hash value, H(m,s), and checks whether it equals the second part of the message, H(m,s). If yes, then Bob believes that the message is legitimate.

- Alice performs the following actions:
  - ❑ concatenates m and s to get (m, s); computes $H(m, s)$
  - ❑ sends $(m, H(m, s))$ to Bob
- Bob computes $H(m, s)$ using m and s; checks whether it equals $H(m, s)$ sent by Alice
- Above approach achieves message integrity without using encryption
- **Terminology**: "Message Integrity" problem also called "Message authentication" problem; s called "*authentication key*" and $H(m, s)$ called "*Message Authentication Code*" (MAC)



It was sent by Alice and was not modified. This shows a block diagram of this protocol. The original message m is concatenated with the secret key s, only known to Alice and Bob, and then the hash value is found. This is the hash value. And then the message m along with the hash value of (m,s) is sent to the receiver.

The receiver takes a message m, and locally appends s (the secret key known only to Bob and Alice), and finds out H(m,s), and checks whether H(m,s) is the same as the hash value sent by the sender. If yes, the verification passes. This approach achieves message integrity without encryption. So, to see why this approach achieves message integrity, notice that if an intruder wants to modify this message m to some other message m', then the intruder must know H(m',s), but the intruder does not know the secret s and hence is not able to find H(m',s). Also, using H(m,s) and without knowing s, it's computationally infeasible to find H(m',s).

Hence, the intruder is not able to replace this message m with another message m'. A note on terminology: the message integrity problem is also called the message authentication problem. This secret s is called the authentication key, and H(m,s) is called the Message Authentication Code, (MAC). So, this is a protocol that is used in many practical protocols used in the internet. To achieve message integrity, we take a message m and append H(m,s), where s is a secret known only to the sender and receiver, and then send (m, H(m,s)). So, we append the message authentication code to the original message and send it.

This is a widely used protocol for achieving message integrity. Thank you.