Reference: Computer Security by Stallings and Brown, Chapter 21

Hashes
Lesson Introduction

In this lesson, we will first introduce the birthday paradox and apply it to decide the length of hash, in order to make hash functions resistant to collision attacks. We will then cover the secure hash function and H max scheme.

- The birthday paradox and length of hash
- Secure hash function
- HMAC

Hash Functions

- Compute message digest of data of any size
- Fixed length output: 128-512 bits
- Easy to compute $H(m)$
- Given $H(m)$, no easy way to find m
  - *One-way function*
- Given m_1, it is computationally infeasible to find $m_2 \neq m_1$ s.t. $H(m_2) = H(m_1)$
  - Weak collision resistant
- Computationally infeasible to find $m_1 \neq m_2$ s.t. $H(m_1) = H(m_2)$
  - Strong collision resistant

A hash function can be applied to a block of data of any size.

A hash function produces an output of fixed size. Typically in the range of 128 bits to 512 bits.

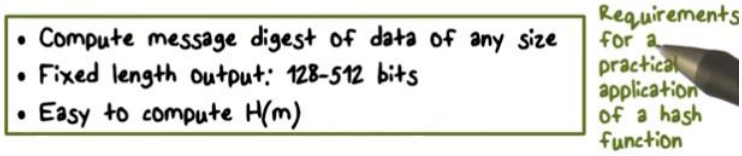
It should be very efficient to compute a hash of a input.

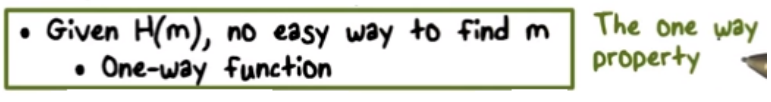
For given hash value it should be computationally infeasible to find the input so that the hash matches the given hash value. That is a hash function should be a one way function.

Given the input data say m1, it should be computationally infeasible to find another input value, say m2 that is not equal to m1. Such that they have the same hash value. This is the weak collision resistant property.

The strong collision resistant property is such that it should be computationally infeasible to five different inputs, m1 not equal to m2 such that they have the same hash value.

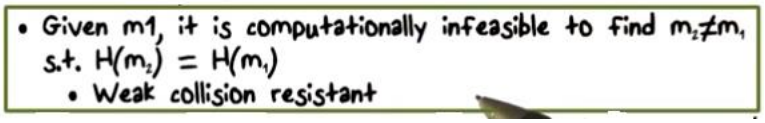
The first three properties make hash functions practical for security applications. In particular, it can handle data of any size, and it's very efficient to compute hash.





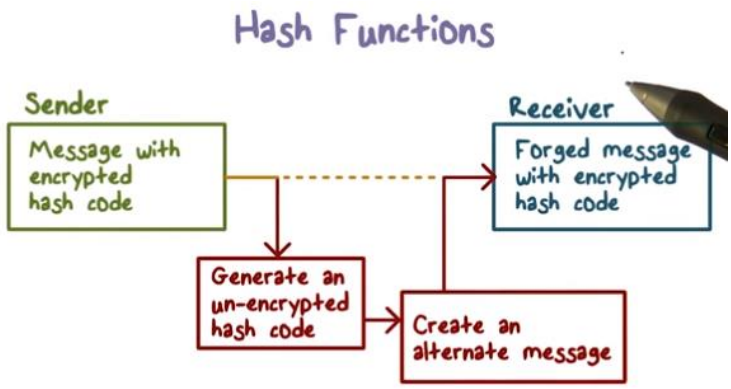
The one way property says that it's quite easy to compute a hash value given a message, but it is virtually impossible to find or generate input message. Given a hash value. This property is very important for

message authentication. For example, we can authenticate a message by hashing a secret value together with the message. The secret is not sent, the hash and the message is sent. If the hash function is not one-way, then the attacker being able to intercept the hash value that's being transmitted. He can then find an input that computes the hash value. And the input would include the secret value, that is the attacker would be able to obtain the secret value. Therefore, this one way property is extremely important.



Hash functions are unique to each message

The weak collision resistant property says that for given message, it is not possible to find a different message, such that it will have the same hash value as the given message. This is a very important property to ensure message integrity.

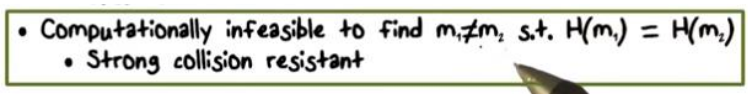


Now let's take a closer look at this weak collision resistant property. Suppose when a sender sends a message to the receiver, he wants to ensure the integrity of the message. So the sender sends a message along with an encrypted hash code of the message.

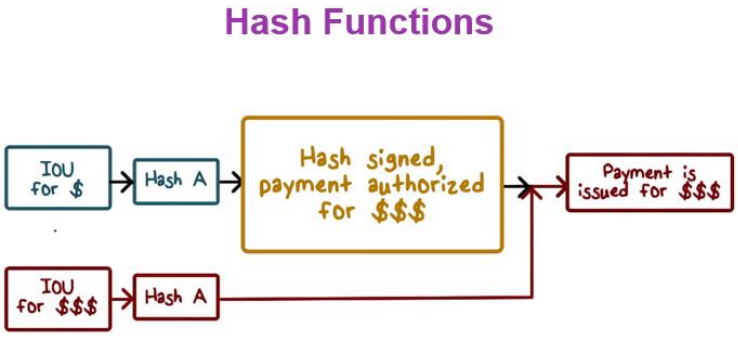
Now an attacker can intercept the message, and the encrypted hash code that is being transmitted. Given a message, the attacker can also compute the hash code. Of course this hash code is not encrypted, because the attacker does not have the key.

Then if the weak collision resistant poverty is not true. The attacker will be able to find another message, such that each hash value is the same as the hash value of the original message.

Then the hacker simply forwards to the receiver the forged message, along with the original encrypted hash code. The receiver will not be able to tell that the message has been modified by the attacker, because the forged message has exactly the same hash value as the original message.



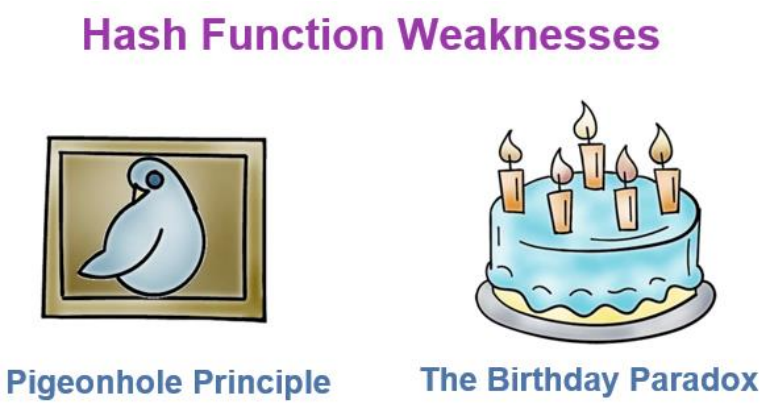
The strong collision resistant property, says that it is not possible to find any pair of two different messages, so that they have the same hash value. It should be obvious that the strong collision resistant property implies the weak collision resistant property. The weak collision resistant property means that the hash function is collision resistant. Only to the specific given input messages. Whereas the strong collision resistant property, means that the hash function is collision resistant to any pair of different messages. Therefore, this is stronger property and it implies the weaker property.



The strong collision resistant property, provides strong message authenticity and integrity protection. Let's take a look at an example.

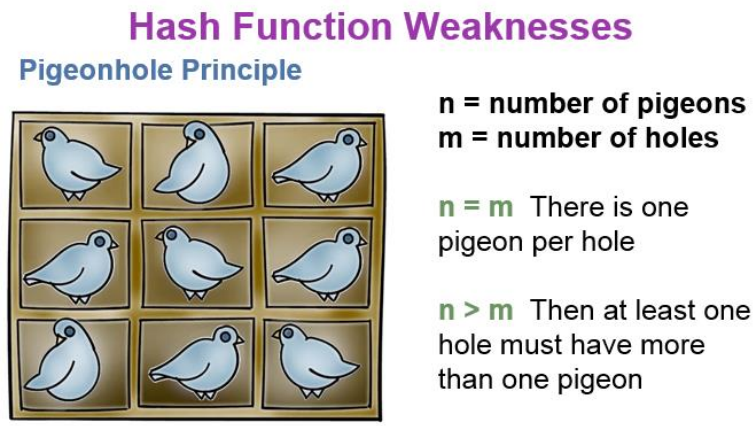
Suppose Bob did some work for Alice, and he wants Alice to pay him later. So he can draft a IOU message and ask Alice to sign and then agree to pay for it. To sign the message, Alice will hash the IOU message. And then sign the hash using her private key. This will prove that Alice authorized the IOU message. In other words, Alice would agree to pay this amount later. In a later day, Bob can present this IOU message, along with the signature, to Alice or Alice's bank to get the money.

Now suppose the strong collision resistant property is not true. That would mean that Bob can find two different messages, one with a smaller dollar amount. For example, this small amount can be just one installment of Alice's payment to Bob. And the larger amount would be several times the amount of money that Alice owes Bob. And if these two messages have the same hash value, then Bob can present the message with a small amount to Alice, and have Alice sign it. Bob can present a signature with a different message. That is, the message with a larger amount, and ask Alice to pay for it. And because these two different messages have the same hash value, Alice can not deny that she has signed it. That means that Alice can not deny that she has agreed to pay this larger amount.



To understand the constraints or potential weaknesses of hash functions, there are two concepts we need to discuss.

The first is the pigeonhole principle and the second is the birthday paradox.



First, let's take a look at the pigeonhole principle. Imagine we have nine pigeonholes, if we have nine pigeons then one pigeon can be placed in each hole. That is suppose we have n number of pigeons and m number of holes. If n equaled m that means there's exactly one pigeon per hole.

Now if we add another pigeon then there will be one hole with two pigeons. That is if n, the number of pigeons is greater than m the number of holes then there's at least one hole that must have more than one pigeon.

Hash Function Weaknesses

The Birthday Paradox

How many people do you need in a room before you have a **greater than 50% chance** that two of them will have the same birthday?

Assume 365 birthdays (our containers)

% chance that two people in the room have the same birthday:

100% requires 366 people (the pigeonhole principle)

Now let's apply the pigeon hole principle to another problem. How many people do we need to have in a room such that there's a good chance that two of them will have the same birthday. There are 365 birthdays and we can think of these birthdays as the pigeon holes. How do we calculate the probability that two people in the room have the same birthday. Obviously, if you want the probability to be 100%, we need 366 people and that's the pigeonhole principle. Because we have 365 birthdays or 365 pigeonholes. So we need 366 pigeons, or people, in order to make sure that two of them will have the same birthday.

Hash Function Weaknesses

The Birthday Paradox

- **Compute probability of different birthdays**
- Random sample of n people (birthdays) taken from *k* (365) days
- k^n samples with replacement
- $(k)_n=k(k-1)\ldots(k-n+1)$ sample without replacement
- **Probability of repetition**:
  - $p = 1- (k)_n/k^n \approx n(n-1)/2k = 0.5$ if $n=\sqrt{k}$

Now suppose, we only need a good chance, say only 50% chance, that two people in the room would have the same birthday. How do we calculate how many people do we need in a room?

We can solve this problem using the following procedure. We can model this problem of having n people choosing from k days as their birthdays. Of course k is 365.

If we allow everybody to choose any of the k days then there are k to the n that many scenarios. Because for each of the n people, they can choose any of the k days. So there are k to the n, that many possible scenarios.

On the other hand, if we insist that no two people should have the same birthday, then the number of scenarios is choosing n out of k without replacement. And that is k times (k-1) times (k- 2), so on and so

forth, and (k- n +1). Because for the first person, he can choose any of the k days. And once he makes his choice, then the next person can only have k- 1 choices, because we don't want the second person to have the same birthday as the first person. And once the first two people have chosen their birthdays, then a third person can only have k-2 choices. And so and so forth. So this is the number of possible scenarios when we insist that no two people should have the same birthday.

Then we can use this formula to compute a probability where there's a scenario where two people share the same birthday. And this can be approximated to n(n-1)/2k. Therefore if you want this probability to be 0.5 meaning that there's 50% chance that two people would share the same birthday then n should be the square root of k. Again this is just an approximation.

- **Probability of repetition**:
  - $p = 1 - (k)_n/k^n \approx n(n-1)/2k = 0.5$ if $n=\sqrt{k}$

Hash Function Weaknesses

The Birthday Paradox

$1-(k)_n/k^n$= **the probability that a pair share the same birthday**

If k = 365, n = 19

If there are **19 people in a room**, there is a good chance that **two of them** share the same birthday!

If k=365 then its square root of k=19. That means as an approximation if we have 19 people in a room, there is a good chance that two of them share the same birthday.

I've tried this in my class every year with the students in my classroom. And it always works out as the math tells us.

Hash Function Weaknesses

Hash Functions:

- There are many more 'pigeons' than 'pigeonholes'
- Many inputs will be mapped to the same output. That is, ***many input messages will have the same hash.***

Conclusion: The longer the length of the hash, the fewer collisions.

Once we understand the pigeonhole principle and birthday paradox, we may realize that some of the properties of hash function seem to contradict each other.

In particular, a hash function can take as input data of any size and the output is always a fixed size. Since the size of hash value is fixed, that means that there are fixed number of possible hash values. On the other hand since the input to hash functions can be of any size that would mean that there are many many more possible inputs to the number of possible outputs. In other words, there are many more pigeons than the pigeonholes.

Then applying the pigeonhole principle, many inputs were mapped to the same output hash value. In other words, many different input messages will have the same hash value, but this violates the property of collision resistance. However, if we take a closer look at hash function properties, in particular, the collision system properties. They only say that it should be compositionally in feasible to find two different messages that have the same hash value. It did not say that it should be mathematically impossible to find such collision.

In other words, although it is mathematically possible to find the collision, the hash function property says that we want to make it invisible, or impractical, to find such collision. So how do we accomplish this? Obviously, the larger the number of possible output hash values, the harder it is to find the collision. In other words, the longer the length of the output hash value, the better. In short, to avoid collision, we should use longer hash values

Hash Length	Possible # of hash values	# of messages to find collision
1	2^1	$2^{1/2}$
64	2^{64}	2^{32}

So, we know that, in order to avoid collision, we should have a longer hash value. Now, the question is, how many bits should we have in a hash value so that it is not feasible to find two different messages that have the same hash value?

Suppose the hash value has L bits. Then, there are 2 to the L, that many possible hash values. According to the birthday paradox we can think about we have 2 to the L, that many possible birthdays, therefore if we have a square root of 2 to the L, that is two to the L divided by two, that many messages. Then there's 50% chance that two of them will have the same hash value. In other words, the attacker only needs to search 2 to the power L divided by 2, that the many messages, in order to find collision.

Therefore, if L is 64, which means the hash value has 64 bits. Then there are 2 to 64, that many possible different hash values, This means that the attacker only needs to search 2 to the 32, that many different messages, in order to find a collision, and this is quite feasible with today's computing power. Therefore, we need the hash value to be longer than 64 bit. Therefore, in hash functions, the hash value is at least 128 bits.

Hash Size Quiz

Choose the correct answer:

If the length of hash is 128 bits, then how many messages does an attack need to search in order to find two that share the same hash?

- ☐ 128
- ☐ 2^{127}
- ☐ 2^{128}
- ☐ 2^{64}

Now let's do a quiz. If the length of hash is 128 bits, then how many messages does an attacker need to search in order to find two that share the same hash value? Is it 128, 2 to the 127, 2 to the 128, or 2 to 64?

Because the length of hash is 128 bits. That means there are 128 possible hash values. Using the birthday paradox, the attacker needs to search a square root of 2 to the 128th, that many possible messages in order to find two that share the same hash. Therefore, the answer is 2 to 64.

- ○ 128
- ○ 2^{127}
- ○ 2^{128}
- ◉ 2^{64}

Secure Hash Algorithm

- **Developed by NIST**, specified in the Secure Hash Standard, originally 1993
- Revised as SHA-1 in 1995
  - 160 bit hash
- **NIST specified SHA2 algorithms in 2002**
  - Hash value lengths of 256, 384, and 512
  - Similar to SHA-1

Now let's discuss the Secure Hash Algorithm or SHA. The Secure Hash Algorithm was developed by NIST.

The original algorithm is SHA-1. SHA-1 produces a hash value of 160 bits. Later, NIST revised the SHA standard and specify the SHA-2 algorithms. This algorithms have key lengths of 256, 384 and 512 bits, and their operations are similar to SHA-1.

This table compares the SHA parameters. The message digest size refers to the length of the hash value, so for SHA-1 it is 160 bits, and for SHA-512 the hash length is 512 bits.

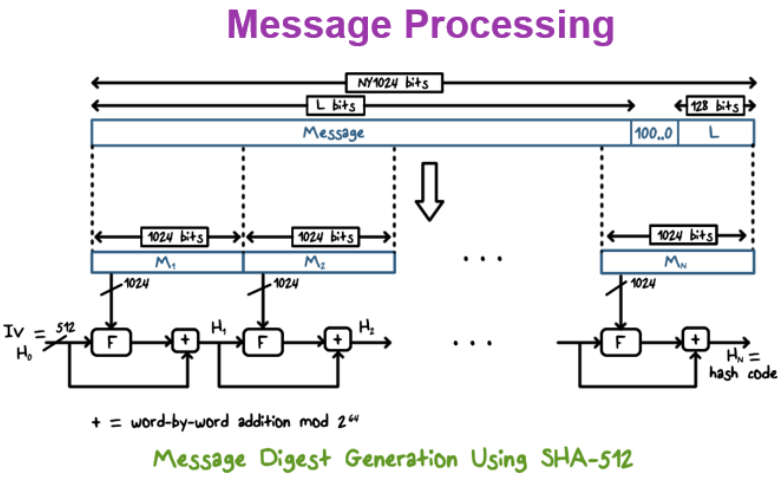
Comparison of SHA Parameters

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	80	80	80
Security	80	128	192	256

Notes:
1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a work factor of approximately $2^{n/2}$.

Message size is a size limit on the input. You can consider these size limits are not having an effect in practice, because most if not all messages will be smaller. For example, even with SHA-1, the message size is limited to 2 to 64. This means that you can hash a message so big that it occupies the entire address space of a 64 bit computer. You can also think about it in another way. 2 to the 43 bits is already a terabyte. As we can see, the algorithms from left to right produce hash values with more bits. Hence, they are more secure as we move from left to right. For example, with SHA-1 because the hash value has 160 bits then the search space for the attacker to find a collision is 2 to 80. Whereas for SHA-512 the search space is 2 to 256.

In this lecture we will provide a description of SHA-512, the other versions are quite similar. For SHA-512 it takes as an input a message that's smaller than 2 to the 128. Again, this is not a limit that will have any impact in practice. The output is a hash value that is in 512 bits. The input is processed block by block and each block has 1024 bits.



Message Digest Generation Using SHA-512

This figure shows the overall processing of a message to produce a hash value.

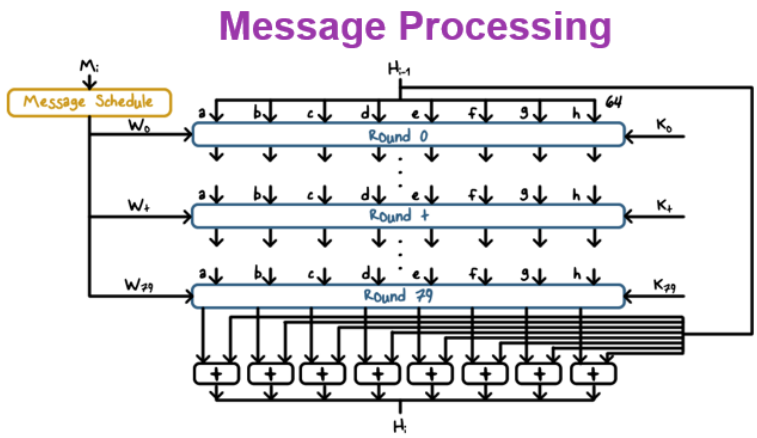
The first step is padding so that the message is a multiple of 1,024 bits because the message is going to be processed block by block. And each block has 1,024 bits. The padding bits are appended to the last block of the message.

First we leave the last 128 bits to store the length of the original message, that is the length before the padding. Then for the space between the original message and the last 128 bits we add one and a number of zeros necessary to fill up the space. And that's how padding works.

After padding, then the message is processed one block at a time. Again, each block has 1,024 bits. The logic of per block processing is described in the next slide. The result of processing the current block is the input to the processing of the next block. That is when processing the second block, the input includes not only the second message block but also the output of the processing of the first block.

And for the first block, the input includes the Iv. The Iv is a 512 bit value, hard coded in the algorithm.

Again the message is processed one block at a time. At each block the input incurs the output of the processing of the previous block. And the result of the processing of the last block is the hash of the entire message.



SHA-512 Processing of a Single 1024-Bit Block

Now let's take a look at the processing of a message block. The processing involves 80 rounds. The input includes not only the current message block but also the result of the processing of the previous block.

And the result of the processing of the current block will be the input to the processing of the next block.

The result from the processing of the previous block is a 512 bit value, and it is divided into 8 64 bit values. And they are a, b, c, d, e, f, g, and h.

Again, the processing involves 80 rounds and each round the input includes the result from the previous round, some constant k, and some words derived from the current message block. The constants here provide randomized values and the purpose is to eliminate any logravities with the input data.

The operations at each round include circular shifts and primitive pulling functions based on and, or, not and x, or. The output of the last round is added to the input to the first round. And the result will be used in the processing of the next message block.

Hash Based Message Authentication

- **Cryptographic hash functions generally execute faster**
- Library code is widely available
- SHA-1 was not designed for use as a MAC because it does not rely on a secret key
- Issued as RFC2014
- **Has been chosen as the mandatory-to-implement MAC for IP security**
- Used in other Internet protocols such as Transport Layer Security (TLS)

Now let's discuss hash based approach to message authentication. Using hash values for message authentication has several advantages.

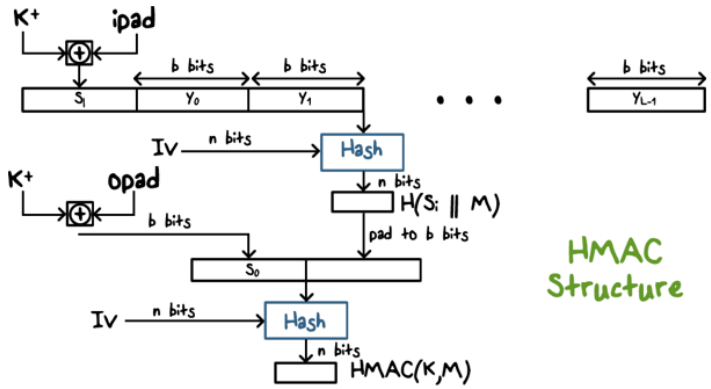
For example, hash functions are very efficient to compute.

And libraries of hash functions are widely available.

On the other hand, a hash function such as SHA cannot be used directly and by itself for message authentication because it does not rely on a secret.

There have been a number of proposals to incorporate a secret key into an existing hash function. The approach that has received the most support is HMAC, and it has been adopted to other protocols such as IPsec and TLS.

Hash Based Message Authentication



Here's an illustration of how HMAC works.

HMAC invokes a hash function and also secret key K.

The message M consists of multiple blocks of b bits, for example, for SHA 512. Each block is 1,024 bits, so b would be 1,024.

So the key K would be first padded to b bits. This is accomplished by simply appending zeros to the end of K.

Then the padded key is XORed with ipad, which is a constant designed to eliminate any irregularities of the key.

And the result is a b bit S1. S1 is prepended to the original message. Then the entire message there is S1, and the original message is hashed to produce a n bit hash value.

For example, if the hash function is SHA 512, then n will be 512. Then the n bit hash value again will be padded to b bits. Then the padded key, K+, is XORed with opad. Again, opad is another constant designed to eliminate irregularities in the key. The result is a b bit value, S0. The padded hash is then appended to S0, and the entire message is hashed. And the n bit result is HMAC of the message with the key, K. And this is how HMAC works. To summarize, HMAC uses an existing hash function and includes a secret key, K, in the processing.

HMAC Security

- Security **depends on the cryptographic strength** of the underlying hash function
- It's much **harder to launch successful collision attacks on HMAC** because of secret key

Let's discuss the security of HMAC.

First of all, the security of HMAC depends on the cryptographic strength of the underlying hash function. That is, the underlying hash function used in HMAC must satisfy the basic properties of one wayness and collision resistant.

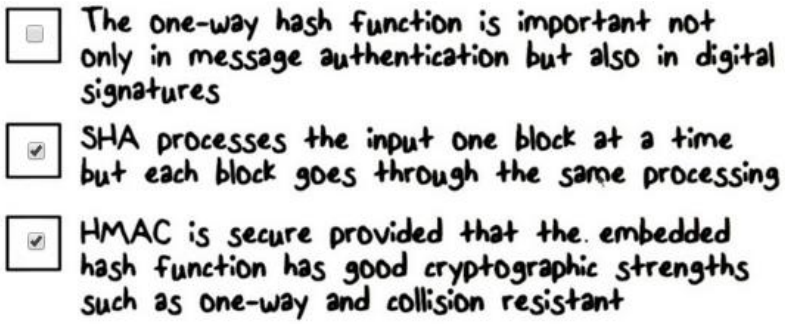
Further, compared with the cryptographic hash function, it is much harder to launch a successful collision attack on HMAC. The main reason is that a secret key is hashed together with the message content. As a result, without knowing the secret key, an attacker cannot compute the correct HMAC. For example, supposed the attacker is able to obtain the HMAC of message M1, and he wants to find another message that have a collision with M1. That is, a different message M2 that's not the same as M1, but had the same HMAC value as M1. But without a secret key, the attacker cannot compute the correct HMAC value for M2. That is, the attacker does not even know whether M1 or M2 will have collision in HMAC. In summary, because of the use of the secret key, HMAC is much more secure than a cryptography hash function.

Hash Function Quiz

Check the statements that are True:

- ☐ The one-way hash function is important not only in message authentication but also in digital signatures
- ☐ SHA processes the input one block at a time but each block goes through the same processing
- ☐ HMAC is secure provided that the embedded hash function has good cryptographic strengths such as one-way and collision resistant

Now let's do a quiz on hash function. Check the statements that are true. First, the one-way hash function is important not only in message authentication, but also in digital signatures. Second, SHA processes the input one block at a time, but each block goes through the same processing. Third, HMAC is secure, provided that the embedded hash function has good cryptographic strengths, such as one-way and collision resistant.



The first statement, the one-way hash function is important not only in message authentication but also in digital signatures. The one-way property of hash function says that it is very efficient to compute a hash value of a given input message. On the other hand, given a hash value, it is computationally infeasible to find the original input message. Hash function can be used in message authentication. For example, if Alice and Bob share a secret key, then Alice can authenticate herself by sending a hello message and hash the hello message along with a share secret key. And when Bob received this hash value, along with a hello message in plain text, Bob can hash the hello message along with a shared key and see whether that hash value matches with the hash value that he just received. And if they match, then Bob knows that this message is from Alice. In other words, Alice is authenticated. Now if the one-way property of hash function does not hold, that means the attacker intercepting the hello message and the hash value can then reverse the hash function and find that the secret key that was used to hash along with the hello message. Therefore, the attacker would able to obtain the share key. Therefore, the one way property of hash function is important for message authentication. Now what about digital signatures? Here's typically how digital signature works. Alice can create a digital signature for message, say m. She first hashes m, and then sign the hash value of m using a private key. Then she sends the message m in plain text along with a signature, therefore the one way property of hash function is not important here because the input plain text message is sent anyway. To recap, this statement is not true because the one way property is important for message authentication but not in digital signatures.

Second statement, SHA processes the input one block at a time but each block goes through the same processing. This is true because this is how SHA works.

Third, HMAC is secure provided that the embedded hash function has good cryptographic strengths, such as one-way, and collision resistant. This is true, that is, if the underlying has function is not secure, the HMAC would not be secure. On the other hand, as we have discussed, HMAC is more secure than a cryptographic hash function because of the use of a secret key.

Hashes
Lesson Summary

- Hash length should be at least 128
  - 2^(64) message to find collision
- SHA1: 160-bit hash; SHA2: 256/384/512-bit hash
  - Message processed/hashed block by block, result to next
- HMAC: hash the message with a secret key
  - Message authentication

In order to resist collision attacks, the length of hash should be at least 128 bits. Secure hash function processes a message block by block and produces an intermediate hash value. HMAC hashes a message with a secret key. It is a standard way to provide message authentication.