# CMSC/Math 456: Cryptography (Fall 2023)

Lecture 20

Daniel Gottesman

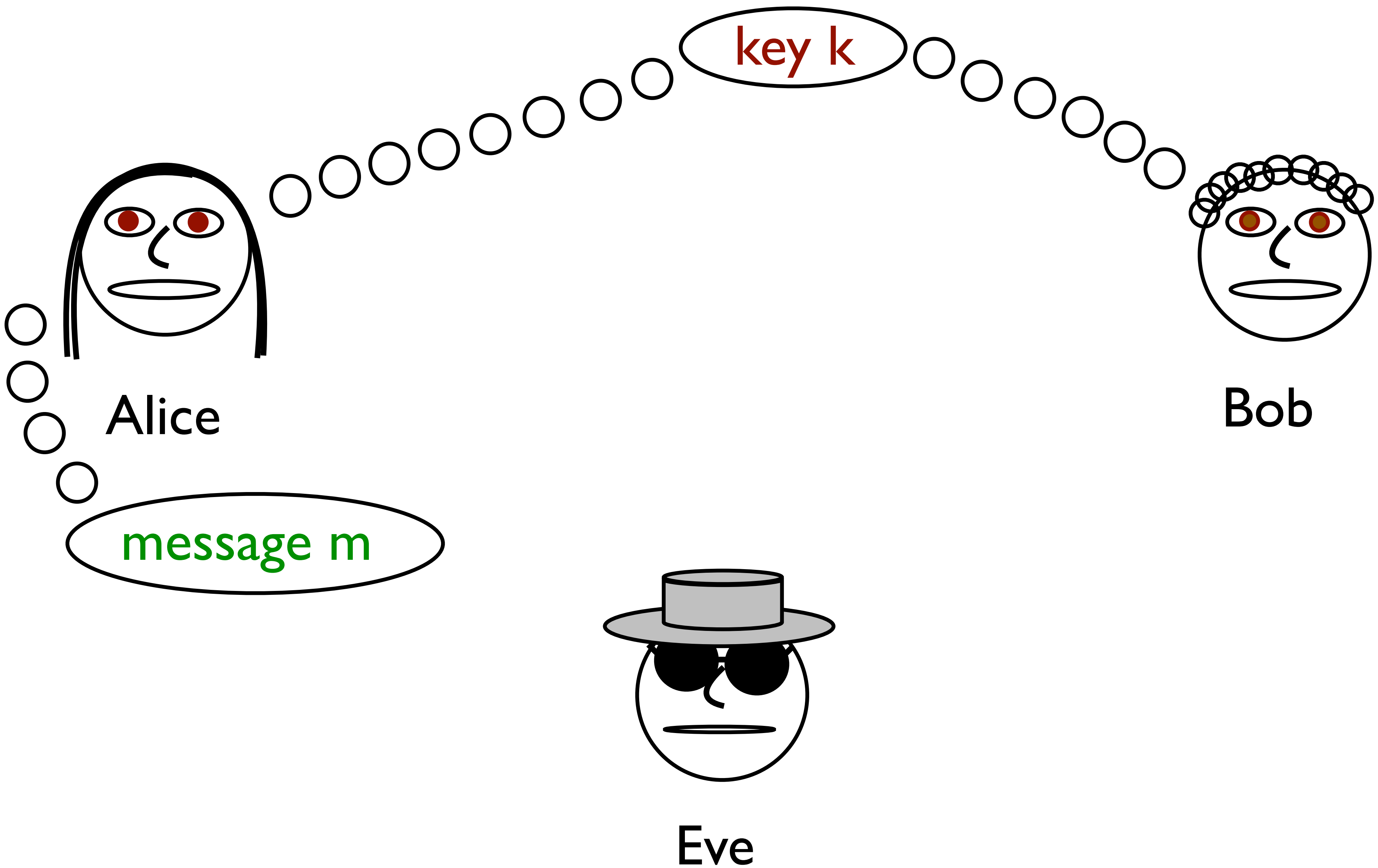
# Administrative

Problem set 7 is due on Thursday at noon.

Grades for the midterm are available. The median score was 88.5. Remember that 100 is the maximum score possible. The original raw scores have been left in for now but anything over 100 will be reduced to 100 before the final grade.

# Message Authentication

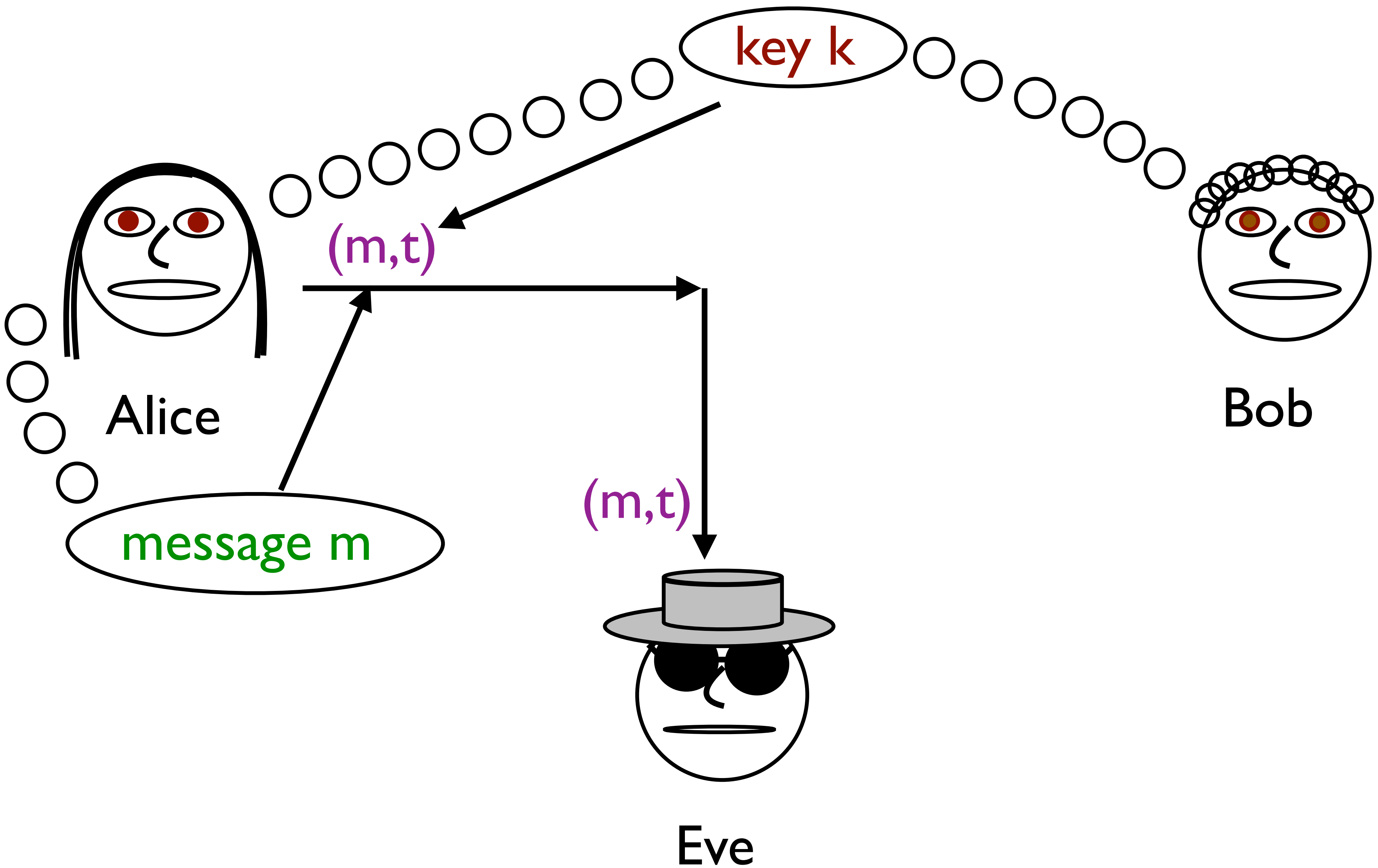
We discussed message authentication, whereby Alice can send a message to Bob without encryption and Bob can be sure it came from Alice unchanged.




This class is being recorded

# Message Authentication

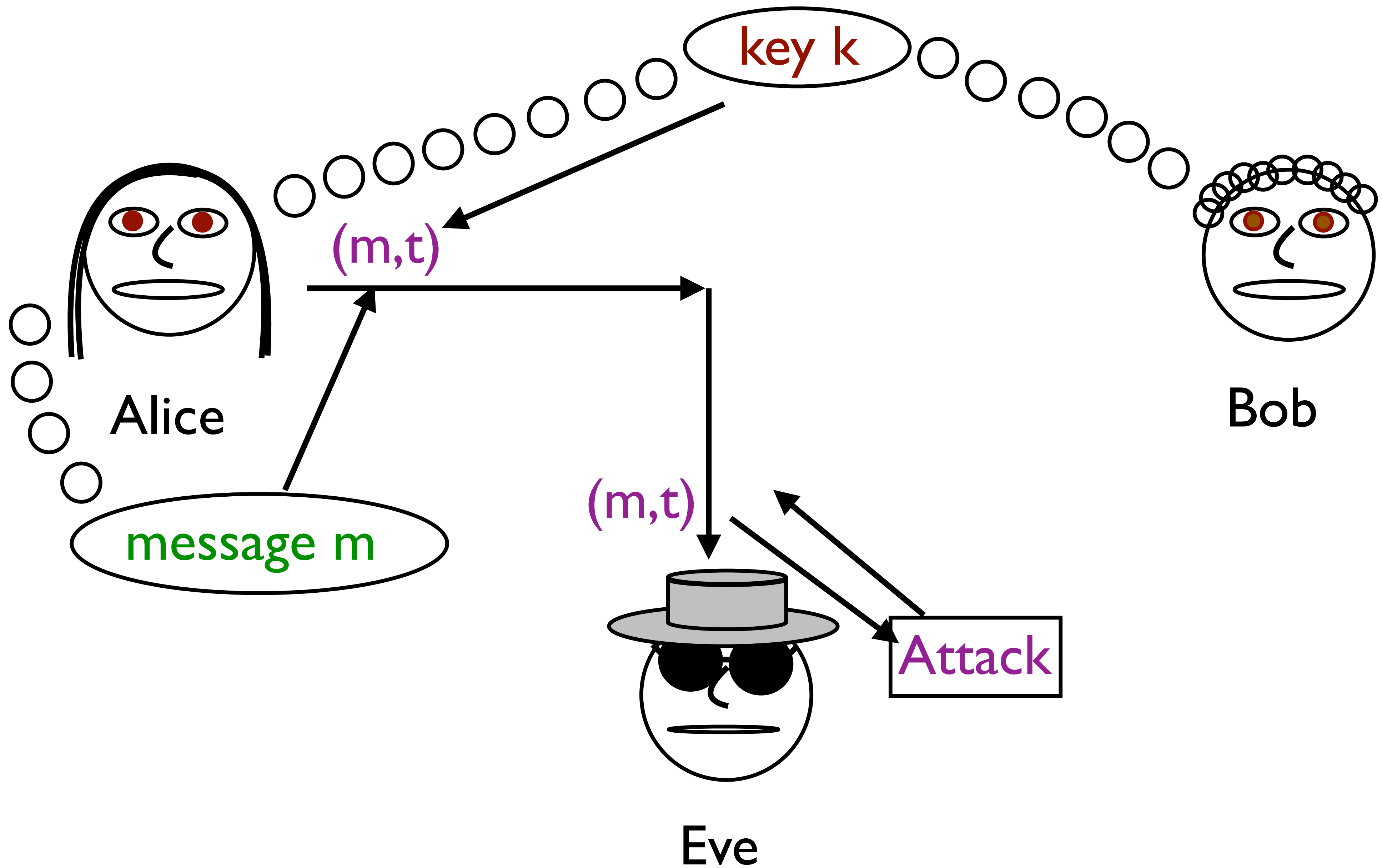
We discussed message authentication, whereby Alice can send a message to Bob without encryption and Bob can be sure it came from Alice unchanged.




This class is being recorded

# Message Authentication

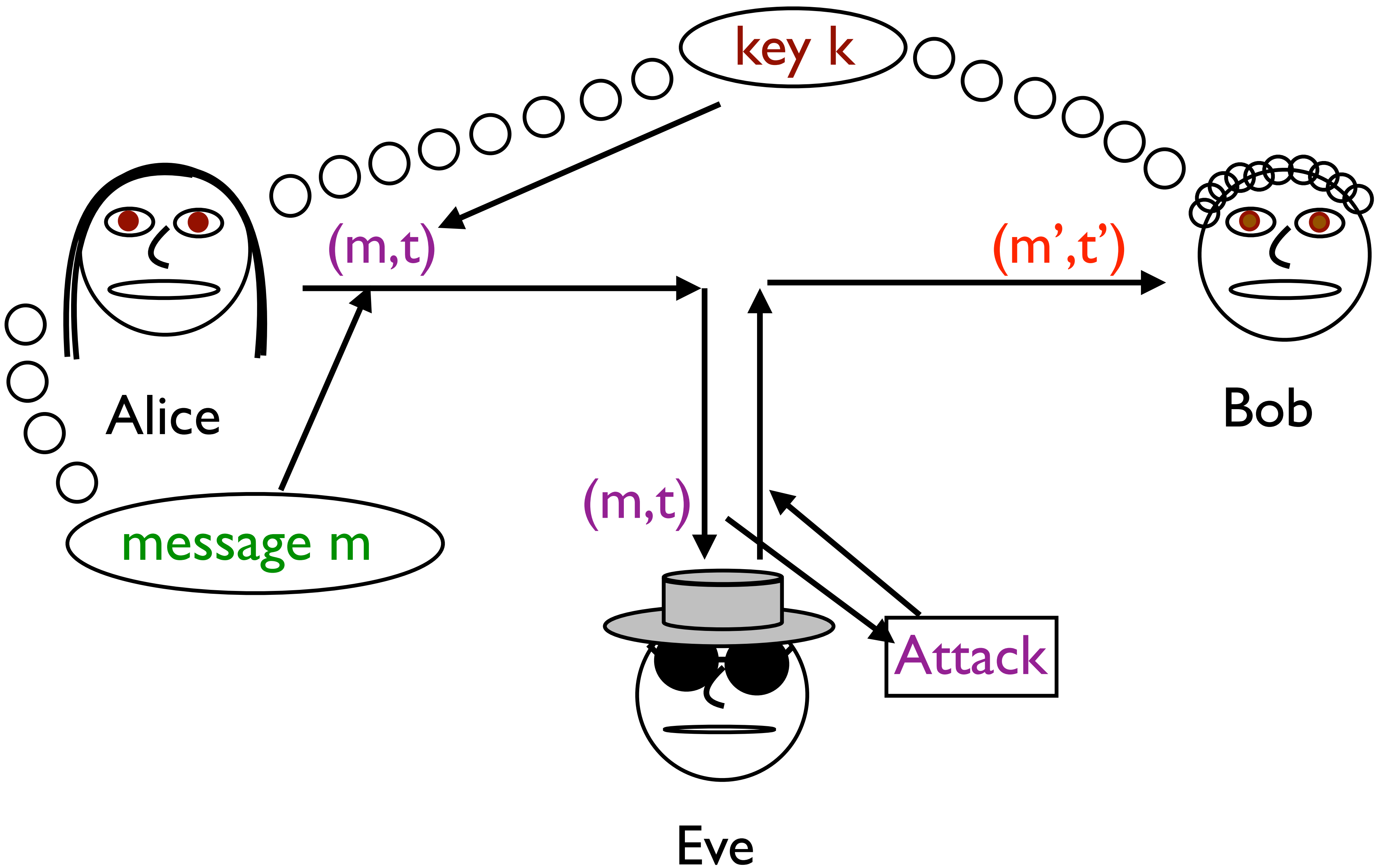
We discussed message authentication, whereby Alice can send a message to Bob without encryption and Bob can be sure it came from Alice unchanged.




This class is being recorded

# Message Authentication

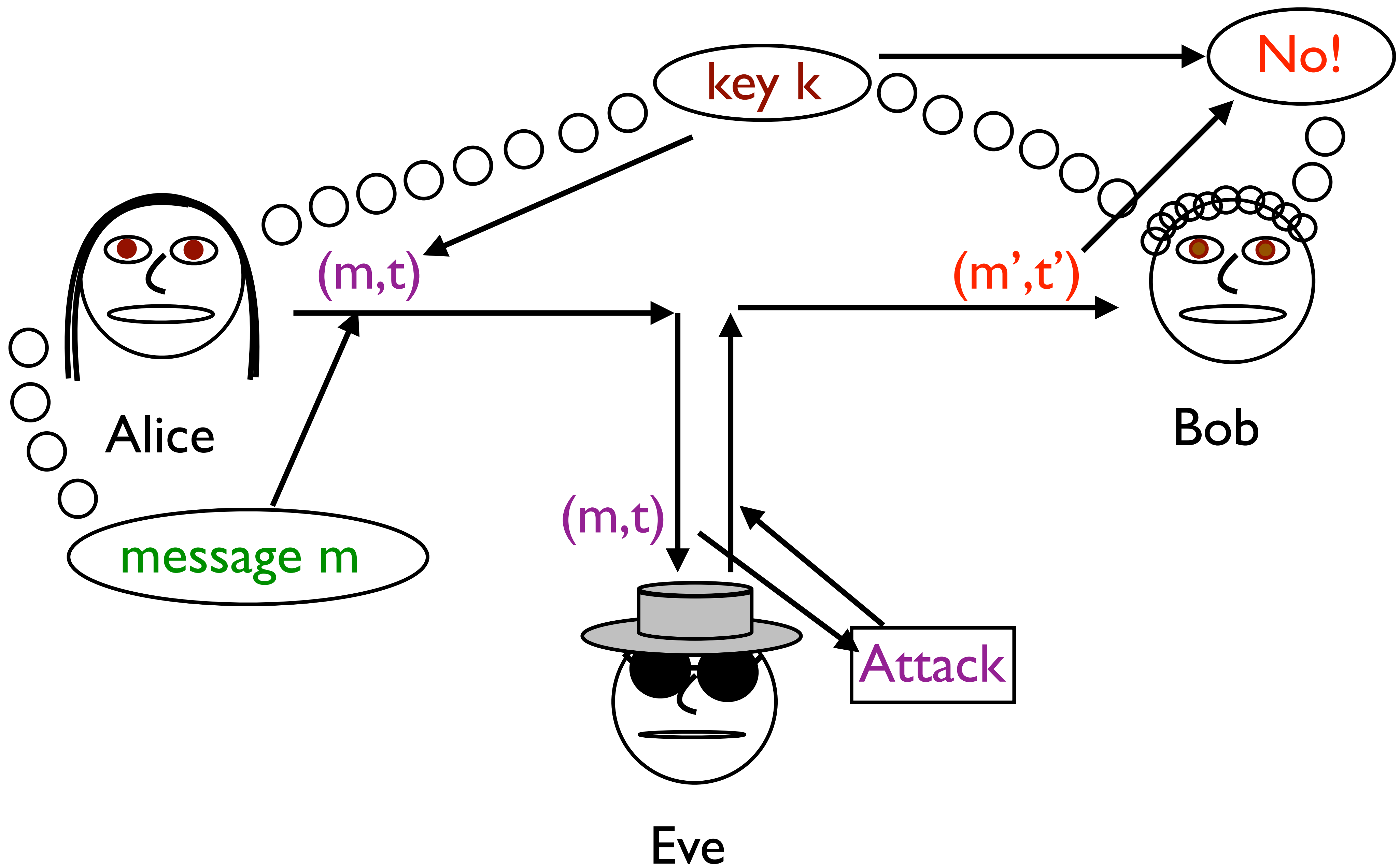
We discussed message authentication, whereby Alice can send a message to Bob without encryption and Bob can be sure it came from Alice unchanged.




This class is being recorded

# Message Authentication

We discussed message authentication, whereby Alice can send a message to Bob without encryption and Bob can be sure it came from Alice unchanged.




This class is being recorded

# MAC Definition

Definition: A message authentication code (MAC) is a set of three probabilistic polynomial-time algorithms (Gen, Mac, Vrfy):

Gen is the key generation algorithm. It takes as input s, the security parameter, and outputs a private key $k \in \{0,1\}^*$ of length poly(s).

Mac is the tag-generation algorithm. It takes as input k and a message $m \in \{0,1\}^*$ and outputs a tag $t \in \{0,1\}^*$.

Vrfy is the verification algorithm. It takes as input k and (m,t) and outputs "valid" or "invalid."

The MAC is correct if

$$\mathrm{Vrfy}(k, m, \mathrm{Mac}(k, m)) = \mathrm{valid}$$

Often Vrfy just runs Mac(k,m) to get a tag t' and outputs "valid" if t=t'.

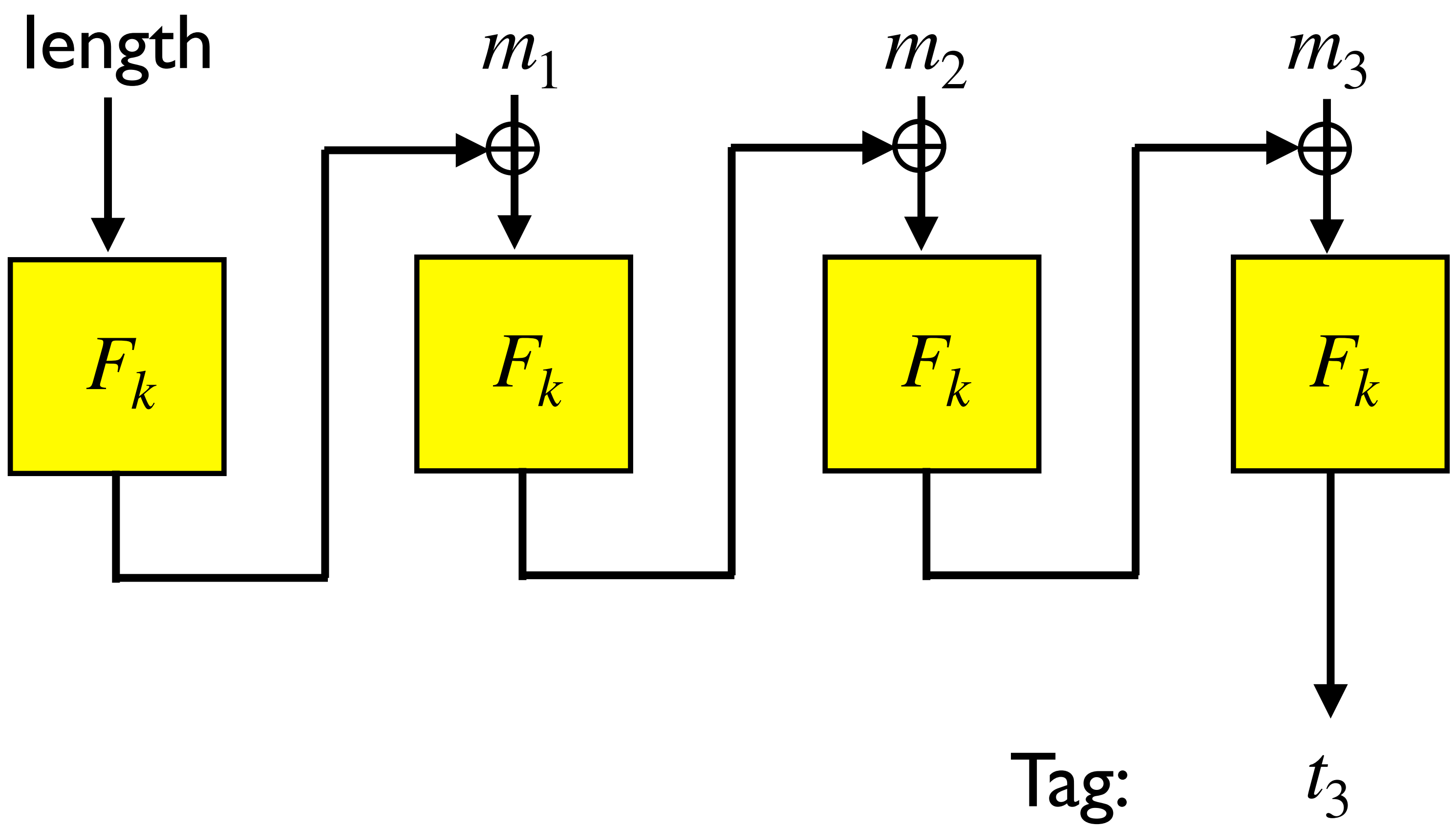
This class is being recorded

# MACs from Pseudorandom Functions

We can make a secure MAC from a pseudorandom function $F_k(m)$. I.e.,

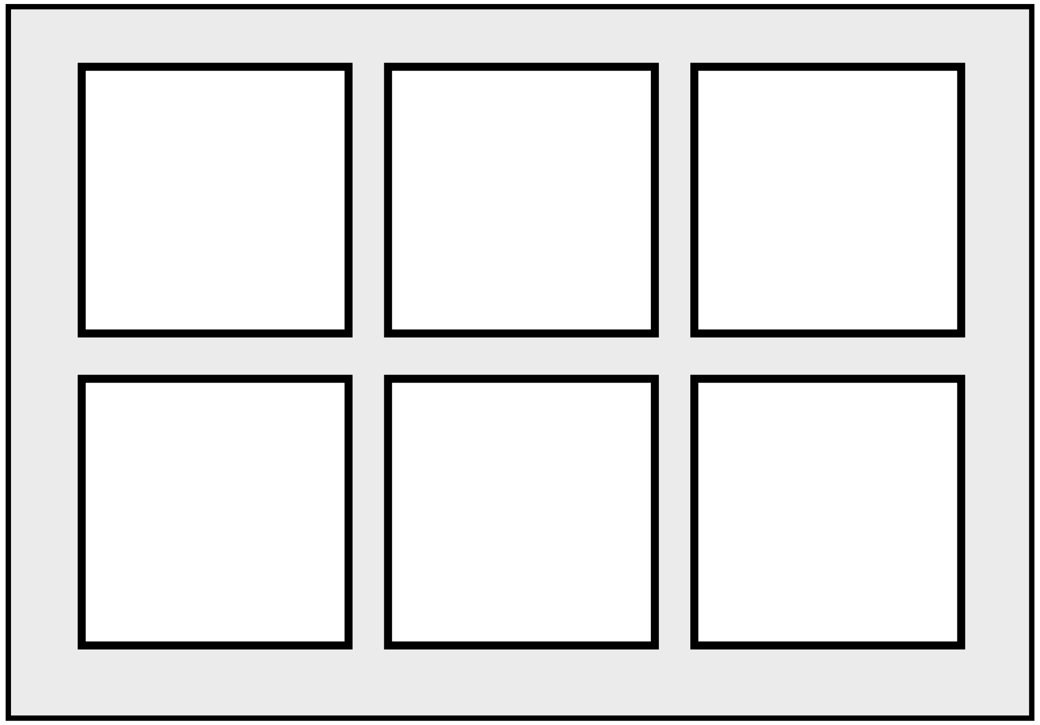
$$\mathrm{Mac}(k, m) = F_k(m).$$

We can also make MACs with a tag much shorter than the message using a CBC-MAC structure:



# Pigeonhole Principle

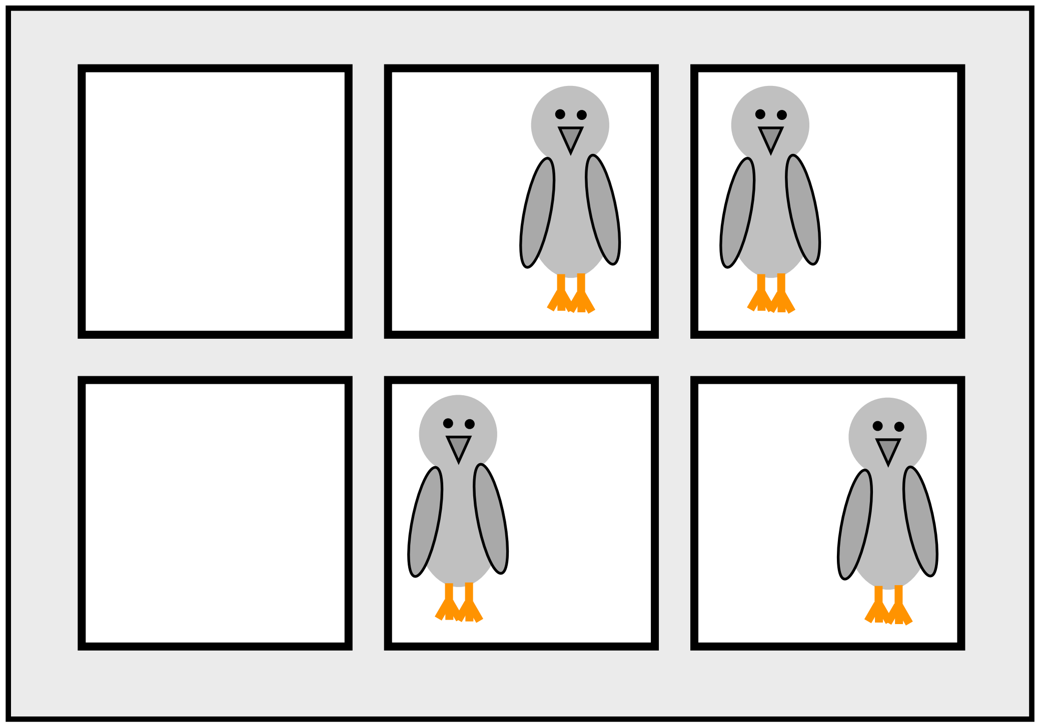
When the tag is shorter than the message, there are more possible messages than tags, which means the pigeonhole principle applies:



Suppose we have n pigeons and m holes.

# Pigeonhole Principle

When the tag is shorter than the message, there are more possible messages than tags, which means the pigeonhole principle applies:

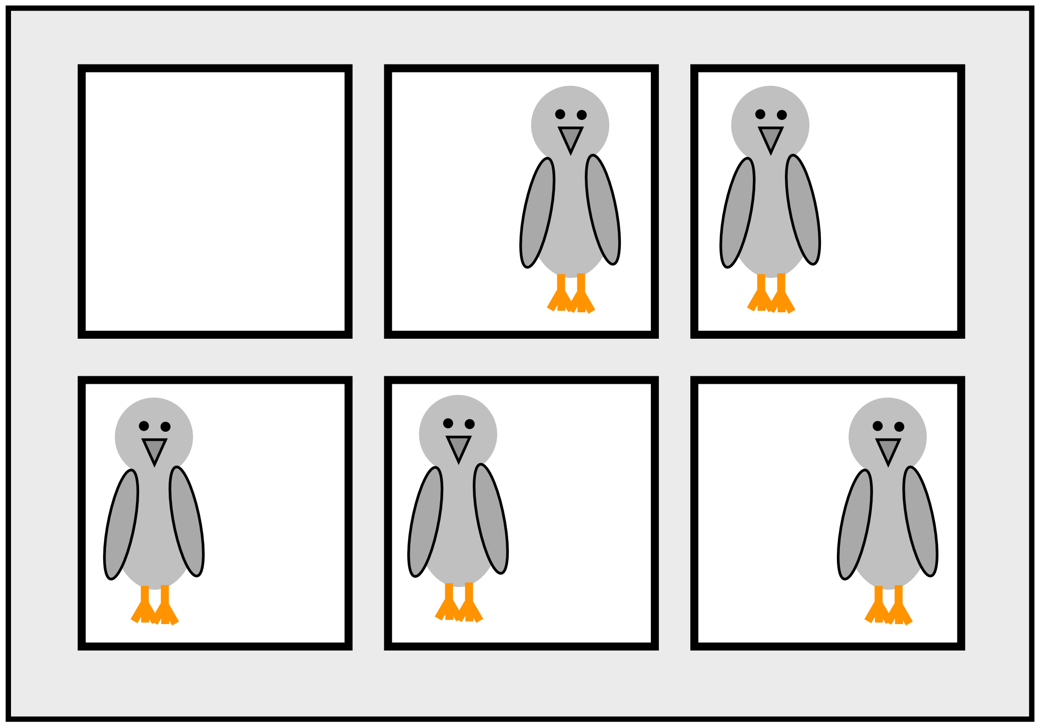


Suppose we have n pigeons and m holes.

If $n \leq m$, then each pigeon can be in its own hole.

# Pigeonhole Principle

When the tag is shorter than the message, there are more possible messages than tags, which means the pigeonhole principle applies:

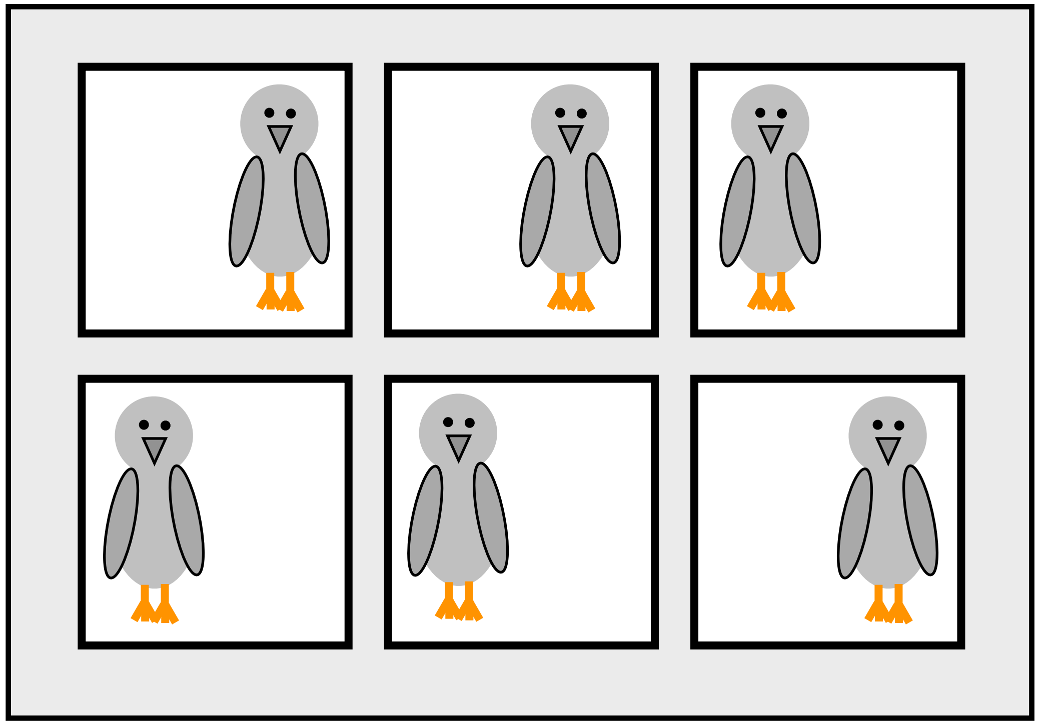


Suppose we have n pigeons and m holes.

If $n \leq m$, then each pigeon can be in its own hole.

# Pigeonhole Principle

When the tag is shorter than the message, there are more possible messages than tags, which means the pigeonhole principle applies:

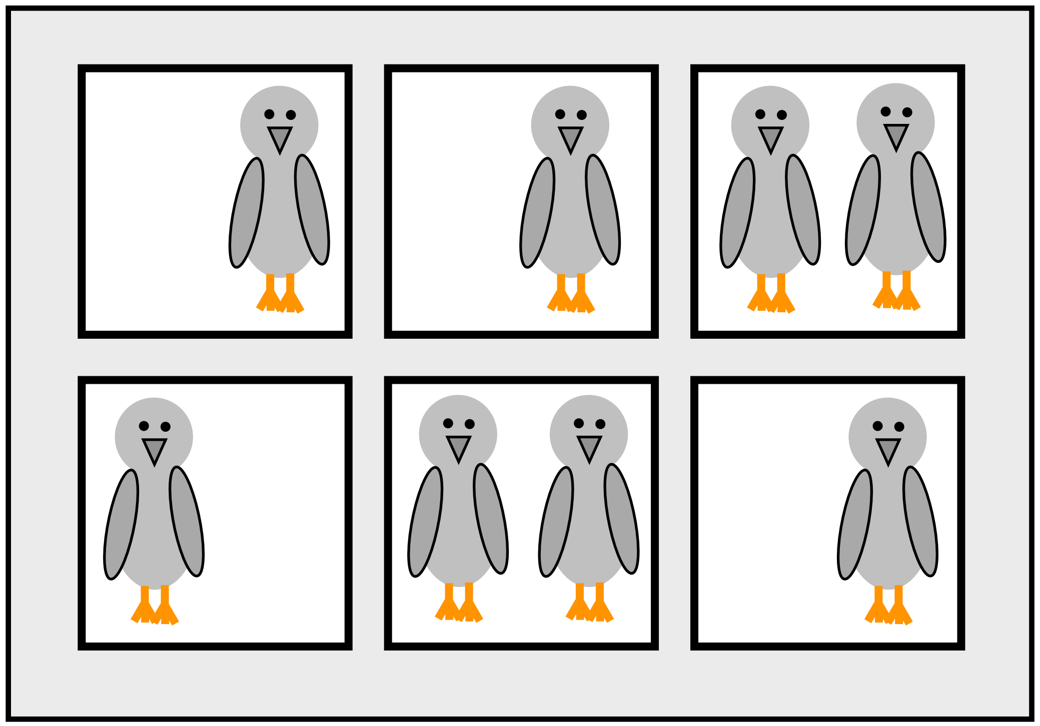


Suppose we have n pigeons and m holes.

If $n \leq m$, then each pigeon can be in its own hole.

# Pigeonhole Principle

When the tag is shorter than the message, there are more possible messages than tags, which means the pigeonhole principle applies:



Suppose we have n pigeons and m holes.

If $n \leq m$, then each pigeon can be in its own hole.

But if $n > m$, then more than one pigeon has to be in the same hole. This is the pigeonhole principle.

# Collisions and MACs

If we want to have long messages and short tags, we are necessarily going to have more than one message that has the same tag. For a MAC to be secure, it should be hard for Eve to find two such messages:

> E.g.: Suppose Eve knows that messages with all bits flipped have the same tag.
>
> Then she sees message 00110011 with tag 1001.
> Now she can forge 11001100 also with tag 1001.

Thus, we need such a MAC to be collision-resistant: It is hard for Eve to find two messages m, m' such that Mac(k,m) = Mac(k,m').

# Hash Functions

Perhaps surprisingly, there are functions that have collision resistance (as far as we can tell) even *without a key* or if the key is known.

A collision-resistant hash function is a function H(x) such that it is hard to find two values x, x' such that H(x) = H(x').

> "Hard" means polynomial-time as usual for us, so to make this a rigorous definition, you actually need to look at a family of hash functions with larger output lengths.
>
> Informally, we can still consider a hash function of fixed size to be collision-resistant if it is hard in practice to find a collision.

This notion only really makes sense if $|H(x)| < |x|$, so the output is shorter than the input.

# Cryptographic Hash Functions

Hash functions are also used in non-cryptographic settings. (E.g. as hash tables.)

> In a non-cryptographic hash function, collisions between typical elements from the domain should be rare.
> In a cryptographic hash function, all collisions should be hard to find.

E.g.: A simple non-cryptographic hash function is mod:

$$H(x) = x \bmod p$$

But it is very easy to find collisions in this function if you are actively trying to.

# Cryptographic Hash Functions

Hash functions are also used in non-cryptographic settings. (E.g. as hash tables.)

In a non-cryptographic hash function, collisions between typical elements from the domain should be rare.
In a cryptographic hash function, all collisions should be hard to find.

E.g.: A simple non-cryptographic hash function is mod:

$$H(x) = x \bmod p$$

But it is very easy to find collisions in this function if you are actively trying to.

Try it: $p = 563$, $x = 822$, $H(x) = 259$

Can you find $x'$ (not 822) with $H(x') = 259$?

# Cryptographic Hash Functions

Hash functions are also used in non-cryptographic settings. (E.g. as hash tables.)

> In a non-cryptographic hash function, collisions between typical elements from the domain should be rare.
> In a cryptographic hash function, all collisions should be hard to find.

E.g.: A simple non-cryptographic hash function is mod:

$$H(x) = x \bmod p$$

But it is very easy to find collisions in this function if you are actively trying to.

Try it: $p = 563$, $x = 822$, $H(x) = 259$

> Can you find $x'$ (not 822) with $H(x') = 259$?

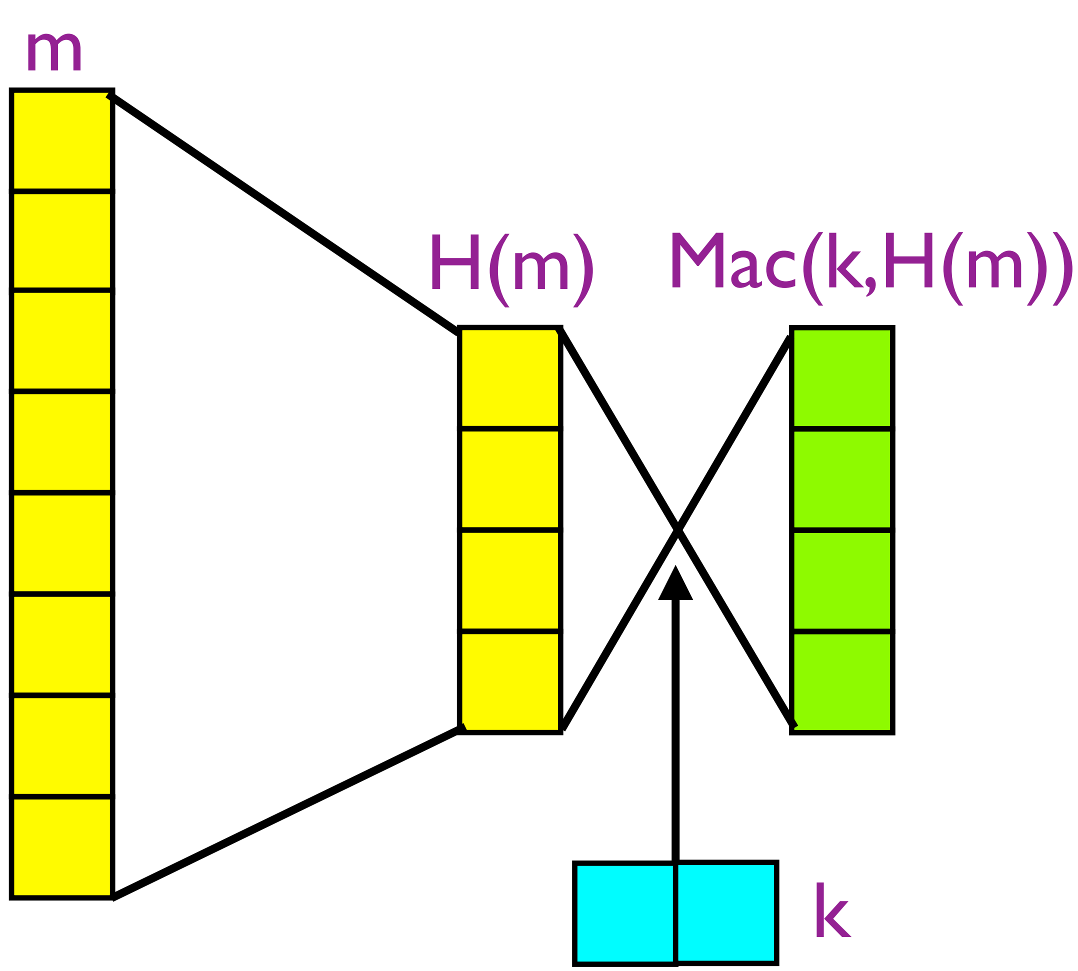
One answer: $x' = 259$

# Hash Functions for MACs

Given a hash function H(x) and a MAC Mac(k,m), we can make a new MAC:

$$\mathrm{Mac}'(k,m) = \mathrm{Mac}(k, H(m))$$

**Theorem:** If H(x) and Mac(k,m) are both secure, then Mac' is also secure.




This allows us to authenticate even long messages using short tags and a small key.

Generally this "hash-and-MAC" protocol is faster than a CBC-MAC because one hash function is easier than many block ciphers.

HMAC is an even better variant of this idea.

# Security of Hash-and-MAC

Suppose Eve is trying to forge a message authenticated with hash-and-MAC.

She has two options to forge (m, MAC(k,H(m))):

# Security of Hash-and-MAC

Suppose Eve is trying to forge a message authenticated with hash-and-MAC.

She has two options to forge (m, MAC(k,H(m))):

- Forge a message with $H(m) \neq H(m')$ for all m' for which she has seen a tag. But that would require breaking the original MAC protocol.

# Security of Hash-and-MAC

Suppose Eve is trying to forge a message authenticated with hash-and-MAC.

She has two options to forge (m, MAC(k,H(m))):

- Forge a message with $H(m) \neq H(m')$ for all m' for which she has seen a tag. But that would require breaking the original MAC protocol.
- Forge a message with $H(m) = H(m')$ for some $m' \neq m$. But that would require breaking the collision-resistance of the hash function.

# Security of Hash-and-MAC

Suppose Eve is trying to forge a message authenticated with hash-and-MAC.

She has two options to forge (m, MAC(k,H(m))):

- Forge a message with $H(m) \neq H(m')$ for all m' for which she has seen a tag. But that would require breaking the original MAC protocol.
- Forge a message with $H(m) = H(m')$ for some $m' \neq m$. But that would require breaking the collision-resistance of the hash function.

Note that both elements are needed.

Try it: Forge a message if the tag is H(m) instead of MAC(k,H(m)).

# Security of Hash-and-MAC

Suppose Eve is trying to forge a message authenticated with hash-and-MAC.

She has two options to forge (m, MAC(k,H(m))):

- Forge a message with $H(m) \neq H(m')$ for all m' for which she has seen a tag. But that would require breaking the original MAC protocol.
- Forge a message with $H(m) = H(m')$ for some $m' \neq m$. But that would require breaking the collision-resistance of the hash function.

Note that both elements are needed.

Try it: Forge a message if the tag is H(m) instead of MAC(k,H(m)).

Answer: Choose any m and compute H(m) — no key needed.

# Security of Hash-and-MAC

Suppose Eve is trying to forge a message authenticated with hash-and-MAC.

She has two options to forge (m, MAC(k,H(m))):

- Forge a message with $H(m) \neq H(m')$ for all m' for which she has seen a tag. But that would require breaking the original MAC protocol.
- Forge a message with $H(m) = H(m')$ for some $m' \neq m$. But that would require breaking the collision-resistance of the hash function.

Note that both elements are needed.

Try it: Forge a message if the tag is H(m) instead of MAC(k,H(m)).

Answer: Choose any m and compute H(m) — no key needed.

Now: Forge a message if H(m) is not collision-resistant.

# Security of Hash-and-MAC

Suppose Eve is trying to forge a message authenticated with hash-and-MAC.

She has two options to forge (m, MAC(k,H(m))):

- Forge a message with $H(m) \neq H(m')$ for all m' for which she has seen a tag. But that would require breaking the original MAC protocol.
- Forge a message with $H(m) = H(m')$ for some $m' \neq m$. But that would require breaking the collision-resistance of the hash function.

Note that both elements are needed.

Try it: Forge a message if the tag is H(m) instead of MAC(k,H(m)).

Answer: Choose any m and compute H(m) — no key needed.

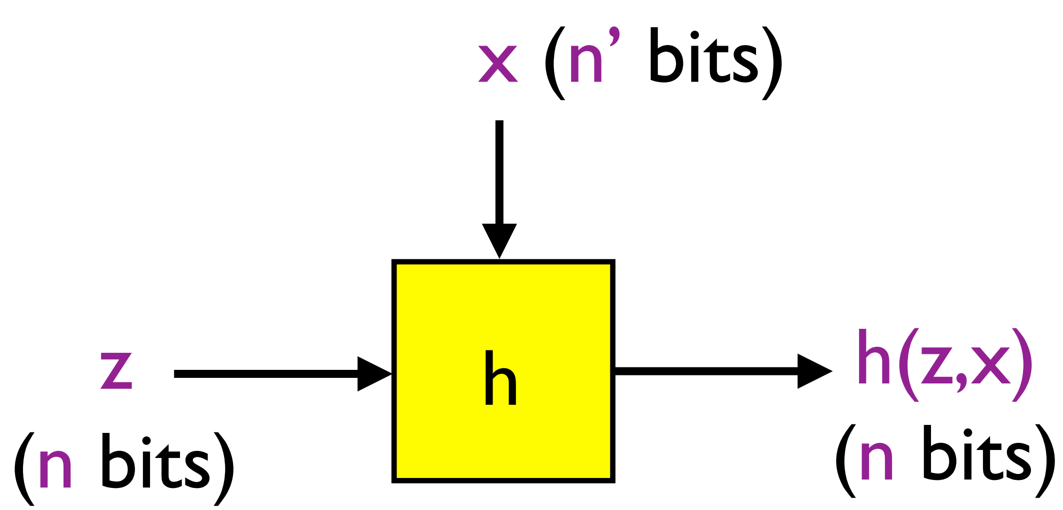
Now: Forge a message if H(m) is not collision-resistant.

Answer: Find a collision $H(m) = H(m')$ and use the tag of m' to forge m. (They have the same tag.)

# Constructing Hash Functions

Strategy: Construct a hash function for fixed-size input blocks. This is called a compression function. Then for larger inputs, break them up into chunks of an appropriate size and apply the compression function repeatedly, adding one chunk each time.

Specifically, we want a compression function h(z,x) that takes two inputs of size n and n', and has one output of size n.



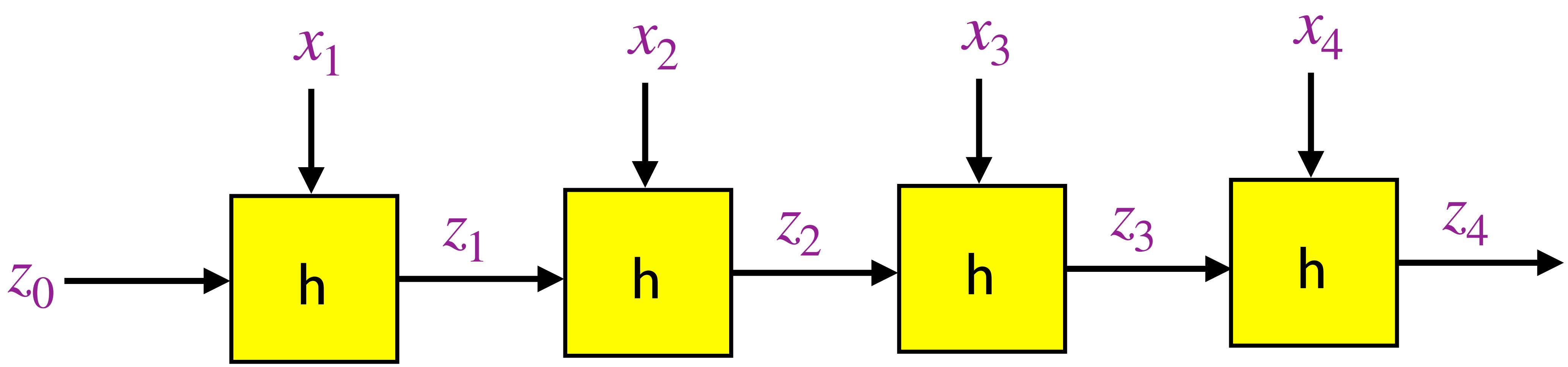

h(z,x) should be collision-resistant.

I.e., it should be hard to find two pairs (z,x) and (z',x') such that $h(z,x) = h(z',x')$.

This class is being recorded

# Merkle-Damgard Construction

To make H(x) for long x:



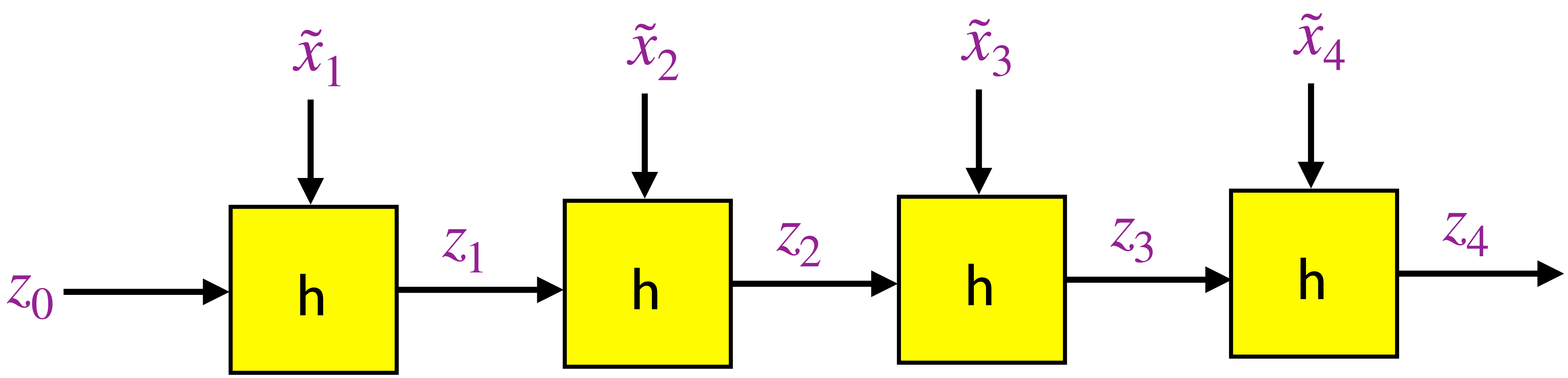

The input x is broken up into $(x_1, x_2, x_3, \ldots)$; each $x_i$ is n' bits long.

$z_0$ is set to some IV. (It is fixed based on the specific construction, not chosen randomly.)

> But … since x might not be a multiple of n', we need to pad it to fill out the blocks. A bad choice of padding makes this construction insecure.

# Padding for Merkle-Damgard

For instance, suppose we pad with 0s. Then an input x that is not a multiple of n' bits long will hash to the same value as $(x \| 0)$.




This class is being recorded

# Padding for Merkle-Damgard

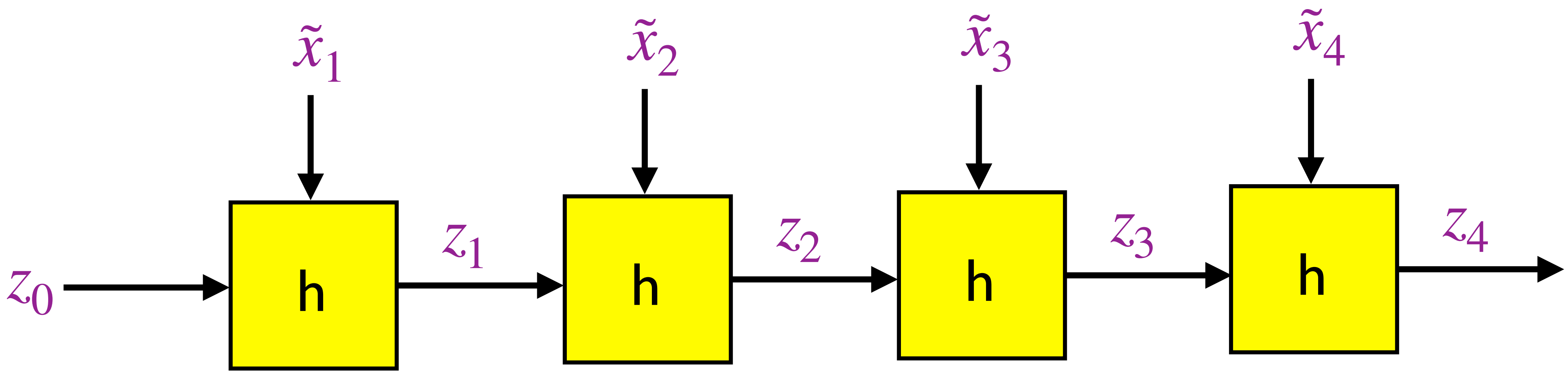
For instance, suppose we pad with 0s. Then an input x that is not a multiple of n' bits long will hash to the same value as $(x||0)$.

## Padding Construction:

Construct $\tilde{x}$ by following x by 1 and then a string of 0s, followed by the length of x. Choose a number of 0s so that the length of $\tilde{x}$ is a multiple of n'. (Length string is always the same size, $\ell \leq n'$ bits long.)

Then break $\tilde{x}$ up into t pieces $\tilde{x}_i$ of n' bits each.

Then $H(x) = z_t$, where $z_i = h(z_{i-1}, \tilde{x}_i)$.




This class is being recorded

# Padding Examples

Examples: Suppose $n' = 6$, $\ell = 3$.

# Padding Examples

Examples: Suppose n' = 6, $\ell$ = 3.

Input x = 1101: Length is 4, written as 100 in binary. Then we will need 2 blocks, total length 12. Thus, we need 5 more padding bits 10000.

$$\tilde{x} = 110110000100,\ \tilde{x}_1 = 110110,\ \tilde{x}_2 = 000100$$

# Padding Examples

Examples: Suppose n' = 6, $\ell$ = 3.

Input x = 1101: Length is 4, written as 100 in binary. Then we will need 2 blocks, total length 12. Thus, we need 5 more padding bits 10000.

$$\tilde{x} = 110110000100,\ \tilde{x}_1 = 110110,\ \tilde{x}_2 = 000100$$

Input x = 1101100: Length is 7, written as 111 in binary. Again we need 2 blocks, total length 12. Thus, we need only 2 more padding bits 10.

$$\tilde{x} = 110110010111,\ \tilde{x}_1 = 110110,\ \tilde{x}_2 = 010111$$

# Padding Examples

Examples: Suppose n' = 6, $\ell$ = 3.

Input x = 1101: Length is 4, written as 100 in binary. Then we will need 2 blocks, total length 12. Thus, we need 5 more padding bits 10000.

$$\tilde{x} = 110110000100,\ \tilde{x}_1 = 110110,\ \tilde{x}_2 = 000100$$

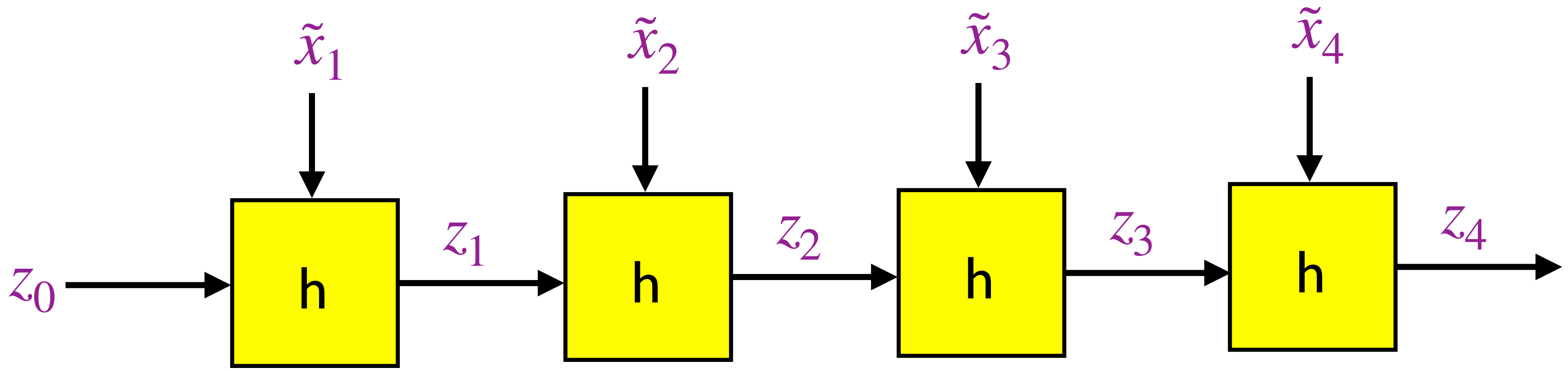
Input x = 1101100: Length is 7, written as 111 in binary. Again we need 2 blocks, total length 12. Thus, we need only 2 more padding bits 10.

$$\tilde{x} = 110110010111,\ \tilde{x}_1 = 110110,\ \tilde{x}_2 = 010111$$

Input x = 10: Length is 2, written as 010 in binary. Now we can manage with 1 block, total length 6. Thus, we need only 1 more padding bit 1.

$$\tilde{x} = 101010,\ \tilde{x}_1 = 101010$$

# Merkle-Damgard security

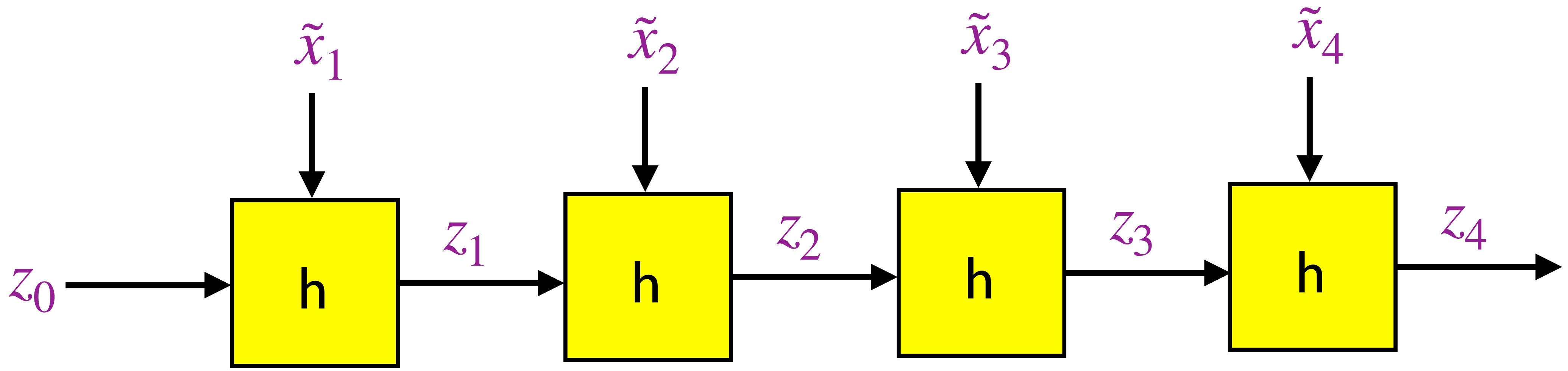



**Theorem:** If $h(z,x)$ is collision-resistant, then $H(x)$ given by the Merkle-Damgard construction is collision-resistant as well.

**Proof Idea:**

This class is being recorded

# Merkle-Damgard security



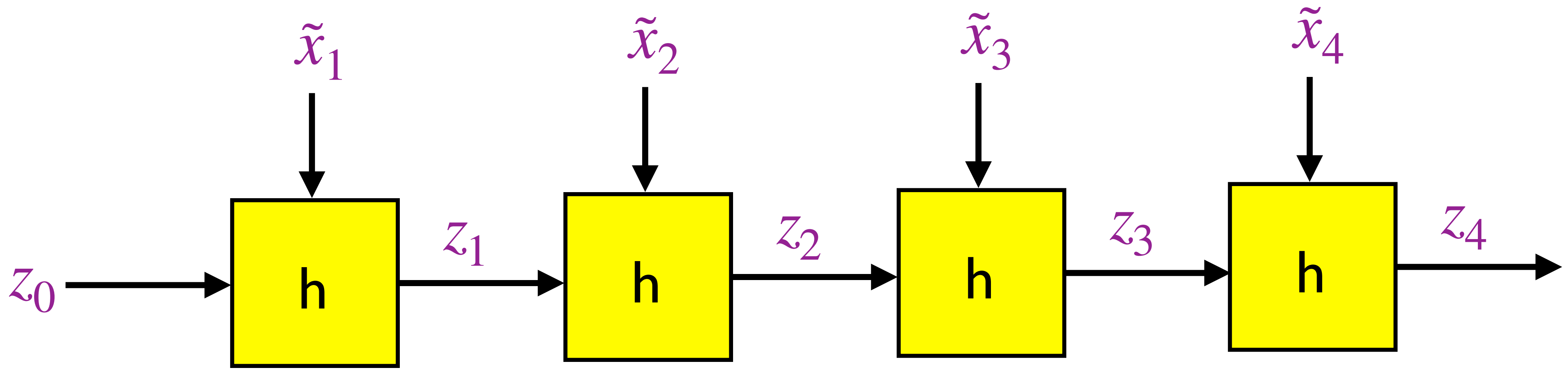

**Theorem:** If h(z,x) is collision-resistant, then H(x) given by the Merkle-Damgard construction is collision-resistant as well.

**Proof Idea:**

Suppose we have two inputs x, x' to H such that $z_i = z_i'$.

This class is being recorded

# Merkle-Damgard security




**Theorem:** If h(z,x) is collision-resistant, then H(x) given by the Merkle-Damgard construction is collision-resistant as well.

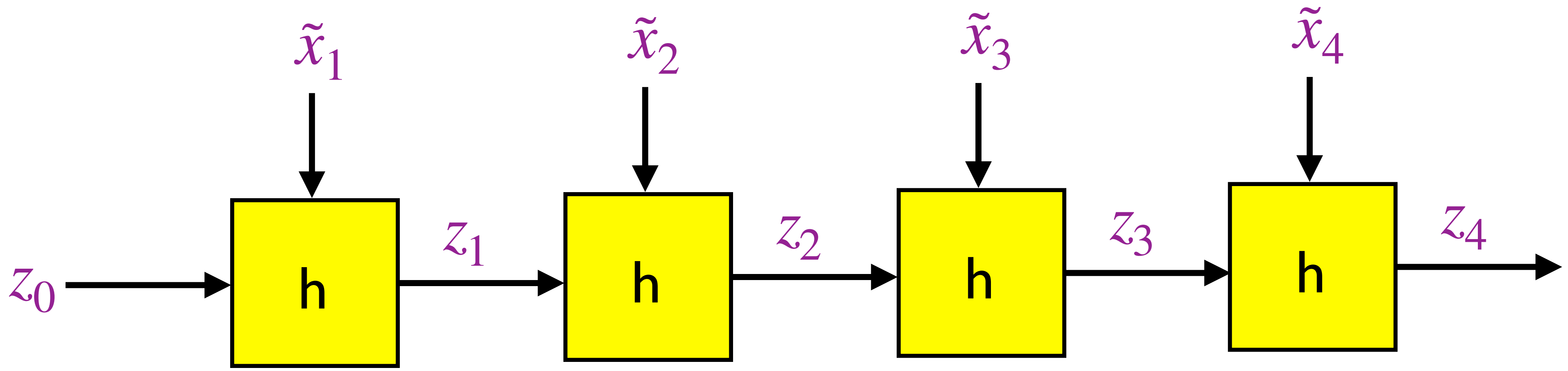
**Proof Idea:**

Suppose we have two inputs x, x' to H such that $z_i = z_i'$.

Recall that $z_i = h(z_{i-1}, \tilde{x}_i) = z_i' = h(z_{i-1}', \tilde{x}_i')$.

This class is being recorded

# Merkle-Damgard security




**Theorem:** If h(z,x) is collision-resistant, then H(x) given by the Merkle-Damgard construction is collision-resistant as well.

**Proof Idea:**

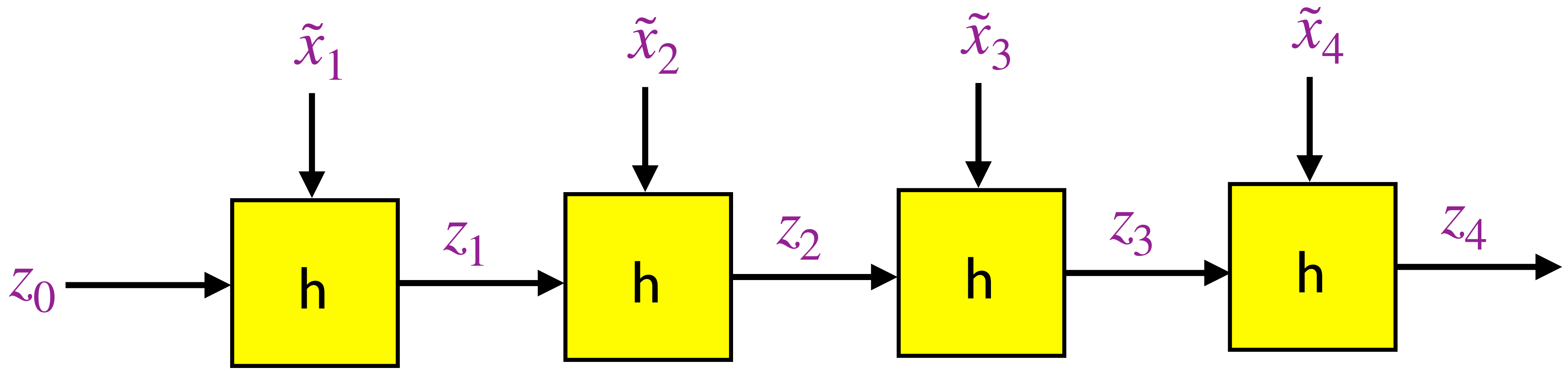
Suppose we have two inputs x, x' to H such that $z_i = z_i'$.

Recall that $z_i = h(z_{i-1}, \tilde{x}_i) = z_i' = h(z_{i-1}', \tilde{x}_i')$.

- If $(z_{i-1}, \tilde{x}_i) \neq (z_{i-1}', \tilde{x}_i')$, then we have a collision in h.

This class is being recorded

# Merkle-Damgard security




**Theorem:** If h(z,x) is collision-resistant, then H(x) given by the Merkle-Damgard construction is collision-resistant as well.

**Proof Idea:**

Suppose we have two inputs x, x' to H such that $z_i = z_i'$.

Recall that $z_i = h(z_{i-1}, \tilde{x}_i) = z_i' = h(z_{i-1}', \tilde{x}_i')$.

- If $(z_{i-1}, \tilde{x}_i) \neq (z_{i-1}', \tilde{x}_i')$, then we have a collision in h.
- Otherwise $z_{i-1} = z_{i-1}'$ and we can apply induction to either find a collision in h or show that x = x'.

This class is being recorded

# Making a Compression Function

A common strategy: the Davies-Meyer construction

Given block cipher $F_k(x)$, let

$$h(z, x) = F_z(x) \oplus x$$

(This doesn't always work. The block cipher F needs to be sufficiently similar to an ideal cipher, which essentially has no exploitable internal structure.)

# Practical Ciphers

Examples of hash functions constructed via Davies-Meyer and Merkle-Damgard:

- MD5 (no longer secure)
- SHA-1 (no longer secure)
- SHA-2 (still OK)

These hash functions use their own block ciphers which involve multiple rounds, in each of which the input is shifted cyclically with some minor changes to most of it and major change to one segment.

- SHA-3 is the newest standard and it is based on different principles.

# Birthday Attacks

Recall the birthday paradox: With N possible dates for birthdays, with high probability, a group of M people has two people with the same birthday if $M \approx \sqrt{N}$.

# Birthday Attacks

Recall the birthday paradox: With N possible dates for birthdays, with high probability, a group of M people has two people with the same birthday if $M \approx \sqrt{N}$.

We can take advantage of this to look for collisions in a hash function: If the hash function H(x) has s-bit output, we will find a collision after computing H on about $2^{s/2}$ random input values.

# Birthday Attacks

Recall the birthday paradox: With N possible dates for birthdays, with high probability, a group of M people has two people with the same birthday if $M \approx \sqrt{N}$.

We can take advantage of this to look for collisions in a hash function: If the hash function H(x) has s-bit output, we will find a collision after computing H on about $2^{s/2}$ random input values.

Consequence: A hash function needs to have output size twice as long as a block cipher to be secure in practice!

# Birthday Attacks

Recall the birthday paradox: With N possible dates for birthdays, with high probability, a group of M people has two people with the same birthday if $M \approx \sqrt{N}$.

We can take advantage of this to look for collisions in a hash function: If the hash function H(x) has s-bit output, we will find a collision after computing H on about $2^{s/2}$ random input values.

Consequence: A hash function needs to have output size twice as long as a block cipher to be secure in practice!

In particular, SHA-2 and SHA-3 have 224-bit and higher output sizes, compared to 128-bit block size and minimum key length for AES.

# Quantum Birthday Attacks

There is a quantum algorithm to find collisions with only $O(2^{s/3})$ function evaluations on a quantum computer. Thus, if quantum computers are a concern, we need to increase hash function sizes further to keep the same security.

Specifically:

# Quantum Birthday Attacks

There is a quantum algorithm to find collisions with only $O(2^{s/3})$ function evaluations on a quantum computer. Thus, if quantum computers are a concern, we need to increase hash function sizes further to keep the same security.

Specifically:

For SHA-2 or SHA-3, 224-bit outputs means a classical computer needs about $2^{112}$ hash function evaluations to find a collision.

# Quantum Birthday Attacks

There is a quantum algorithm to find collisions with only $O(2^{s/3})$ function evaluations on a quantum computer. Thus, if quantum computers are a concern, we need to increase hash function sizes further to keep the same security.

Specifically:

For SHA-2 or SHA-3, 224-bit outputs means a classical computer needs about $2^{112}$ hash function evaluations to find a collision.

A quantum computer would need only about $2^{224/3} < 2^{75}$ hash function evaluations.

# Quantum Birthday Attacks

There is a quantum algorithm to find collisions with only $O(2^{s/3})$ function evaluations on a quantum computer. Thus, if quantum computers are a concern, we need to increase hash function sizes further to keep the same security.

Specifically:

For SHA-2 or SHA-3, 224-bit outputs means a classical computer needs about $2^{112}$ hash function evaluations to find a collision.

A quantum computer would need only about $2^{224/3} < 2^{75}$ hash function evaluations.

A hash function with 336-bit outputs would need $2^{112}$ hash function evaluations on a quantum computer, and could thus have post-quantum security comparable to SHA-2 or SHA-3 vs. classical attacks.

# Key Derivation

Recall in the RSA-based KEM, we needed to pick a key derivation function to convert the key to bits. We also needed it with Diffie-Hellman.

RSA KEM:

Gen: Pick a (public) key derivation function H(x), then as usual for RSA, i.e., generate two random primes p and q which are s bits long. Let N = pq. Choose $e, d \in \mathbb{Z}_N^*$ s.t. $ed = 1 \bmod \varphi(N)$. The public key is (N, e) and the private key is (N, d).

Encaps: Choose random x. The ciphertext is $c = x^e \bmod N$ and the key is H(x).

Decaps: Given c and d, compute $x' = c^d \bmod N$. Then the key is H(x').

# Hash Functions for Key Derivation

One solution (although not the only one) is to use a hash function for key derivation.

Public key output

Actual key

$k \longrightarrow H(k)$

We don't care so much about collisions here. Instead what we want is that $H(k)$ should be roughly uniformly distributed and that any partial information Eve might have about $k$ should get squeezed out, so she has little information about $H(k)$.

> These come from a stronger property of the hash function, namely that it can be well modeled by a random oracle.

# Hash Functions as Random Oracles

A random oracle is a random function which is implemented as a black box. That is, we count complexity by how many times the function is evaluated.

> If H(x) is a random oracle, knowing the value of $H(x_i)$ for any set $\{x_i\}$ doesn't help you predict H(y) if $y \notin \{x_i\}$.

# Hash Functions as Random Oracles

A random oracle is a random function which is implemented as a black box. That is, we count complexity by how many times the function is evaluated.

> If H(x) is a random oracle, knowing the value of $H(x_i)$ for any set $\{x_i\}$ doesn't help you predict H(y) if $y \notin \{x_i\}$.

A random oracle is automatically collision resistant:

# Hash Functions as Random Oracles

A random oracle is a random function which is implemented as a black box. That is, we count complexity by how many times the function is evaluated.

> If H(x) is a random oracle, knowing the value of $H(x_i)$ for any set $\{x_i\}$ doesn't help you predict H(y) if $y \notin \{x_i\}$.

A random oracle is automatically collision resistant:

> For any pair H(x), H(y) ($x \neq y$), the probability that $H(x) = H(y)$ is exactly $2^{-s}$, where H has an s-bit output. This is precisely the limit on collisions set by birthday attacks.

# Hash Functions as Random Oracles

A random oracle is a random function which is implemented as a black box. That is, we count complexity by how many times the function is evaluated.

> If H(x) is a random oracle, knowing the value of $H(x_i)$ for any set $\{x_i\}$ doesn't help you predict H(y) if $y \notin \{x_i\}$.

A random oracle is automatically collision resistant:

> For any pair H(x), H(y) ($x \neq y$), the probability that $H(x) = H(y)$ is exactly $2^{-s}$, where H has an s-bit output. This is precisely the limit on collisions set by birthday attacks.
>
> > But collision resistance doesn't necessarily imply that the hash function is a good random oracle.

# Random Oracle vs. Pseudorandom

A random oracle can be used to make a pseudorandom generator or function, but they are not the same.

**Random Oracle**

- No key
- Can be evaluated by everyone
- Outputs always look random

**Pseudorandom Function**

- Keyed
- Evaluated only using the key
- Outputs look random only if key is unknown.

Both are things that *look like* random functions, but in different contexts.

# What Does it Mean to be RO?

When we say a hash function is "well-modeled as a random oracle," we mean that it has no *useful* structure for an attacker to exploit.

It means

- The output strings have no special properties and no discernible correlations.
- Birthday attacks are the best or only way to find collisions.
- Computing intermediate values (i.e., stopping the evaluation partway through) is not useful for speeding up searches.
- In general, brute force attacks are the best or only attacks that will work.

Any real function must fail to match a random oracle to some degree, but some functions seem to be close enough in practice for security proofs based on random oracles to be useful.

# Key Derivation and Random Oracle

Public key output

Actual key

$$k \longrightarrow H(k)$$

If H(k) is well-approximated as a random oracle, then it is a good key derivation function:

- All output strings are equally likely, averaged over H.
- If Eve has narrowed k down to N possibilities, there are still N possibilities for H(k), but they are completely unrelated to each other. E.g., even if Eve knows the first few bits of k, she does not know any bits of H(k).