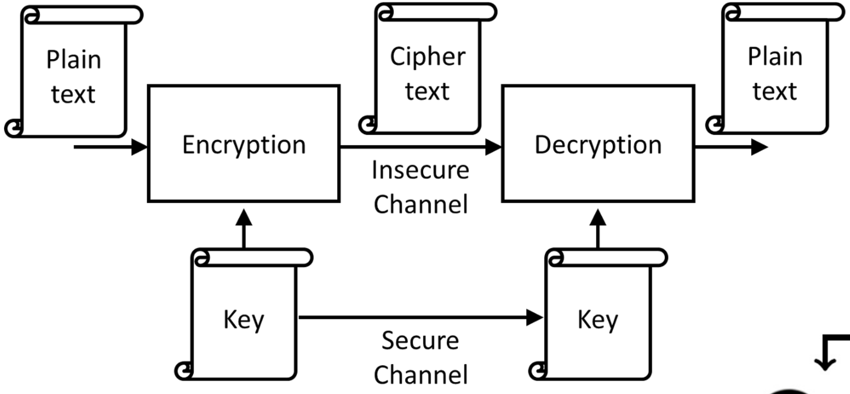
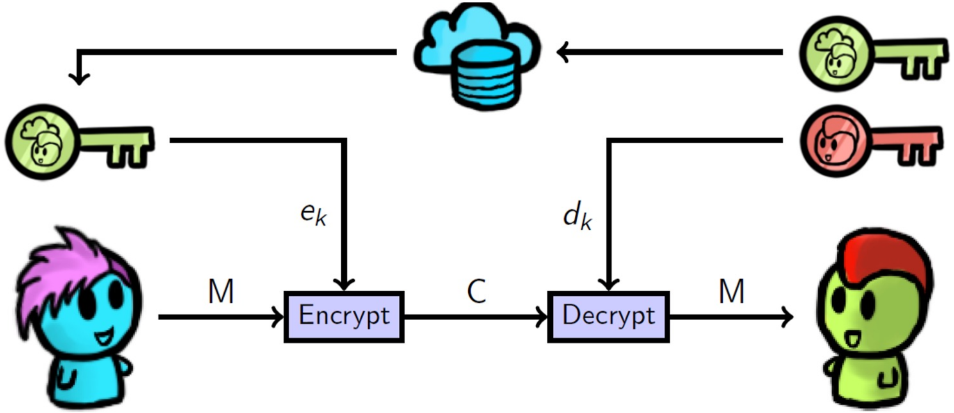
# CS489/698 Privacy, Cryptography, Network and Data Security

Integrity and Authenticated Encryption

Spring 2024, Monday/Wednesday 11:30am-12:50pm

# Block/Stream Ciphers, Public Key Cryptography…



# Size of message on textbook RSA

- Overview:

$$(x^e)^d \equiv x \bmod N$$

# Size of message on textbook RSA

- Overview:

$$(x^e)^d \equiv x \bmod N$$

**x** has to be strictly smaller than **N**, otherwise decryption will produce erroneous values.

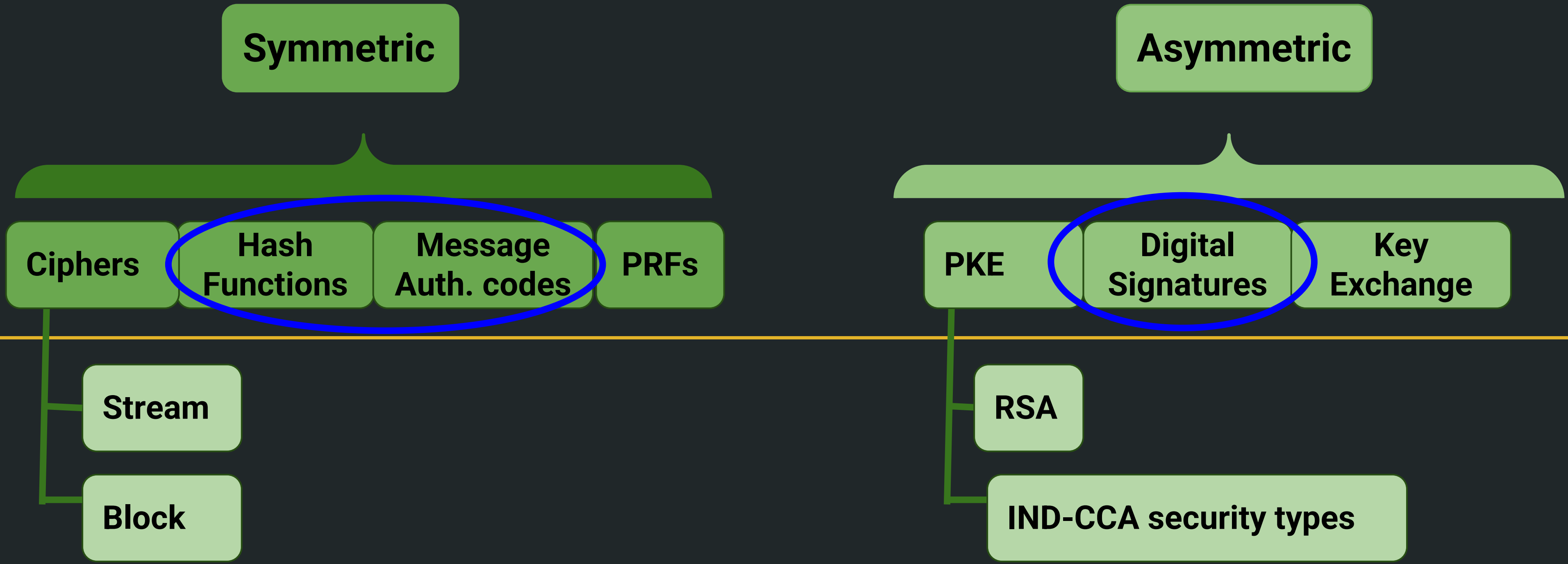
# Size of message on textbook RSA

- Overview:

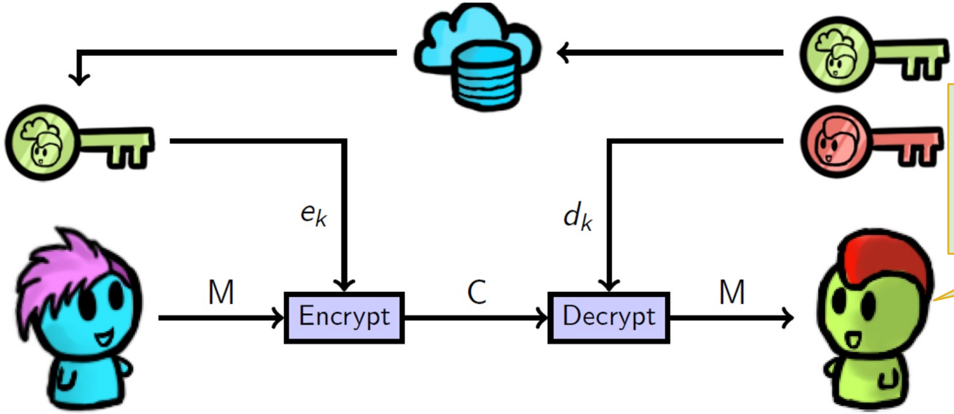
$$(x^e)^d \equiv x \bmod N$$

Ok! So we can break the message in **chunks**! But perhaps we're better served with **hybrid** schemes…

**x** has to be strictly smaller than **N**, otherwise decryption will produce erroneous values.

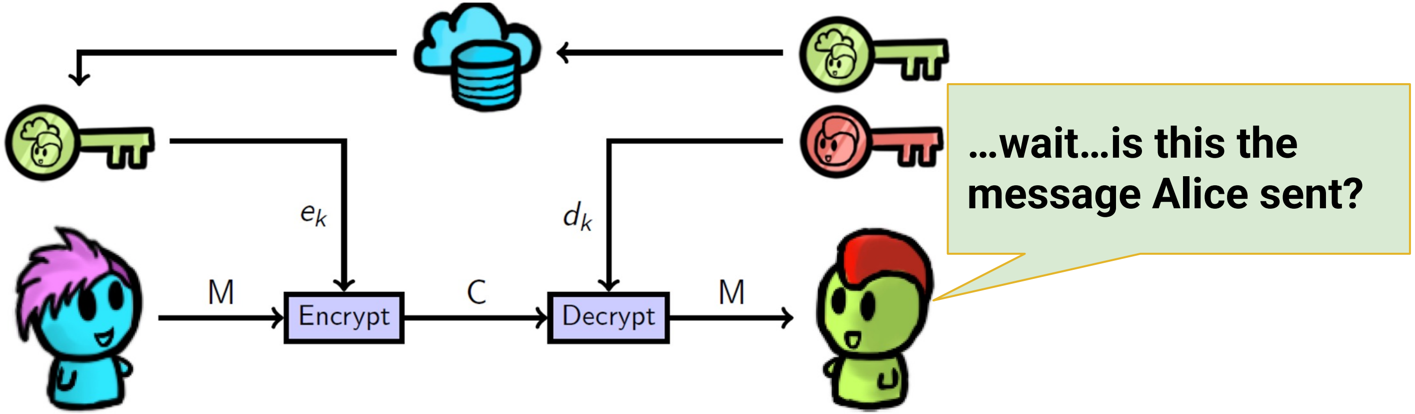

Symmetric
Asymmetric
Ciphers
Hash Functions
Message Auth. codes
PRFs
PKE
Digital Signatures
Key Exchange
Stream
Block
RSA
IND-CCA security types

# Can we Detect Messages Changed in Transit?




…wait…is this the message Alice sent?

# Can we Detect Messages Changed in Transit?





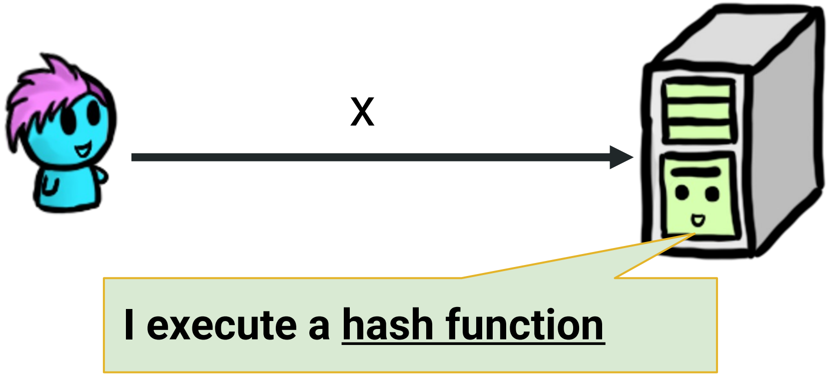
**Checksums**, appended so Bob can verify it

# Not. Good. Enough.

**Checksums are deterministic…I can construct fake ones.**

**Goal:** Make it hard for Mallory to find a second message with the same checksum as the “real” one

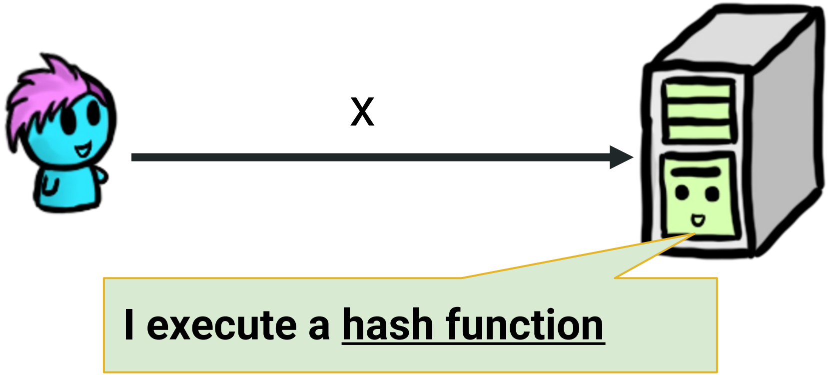
# Towards Integrity: Cryptographic Hash Functions




Common examples:

- MD5, SHA-1, SHA-2, SHA-3 (aka Keccak after 2012)

# Towards Integrity: Cryptographic Hash Functions

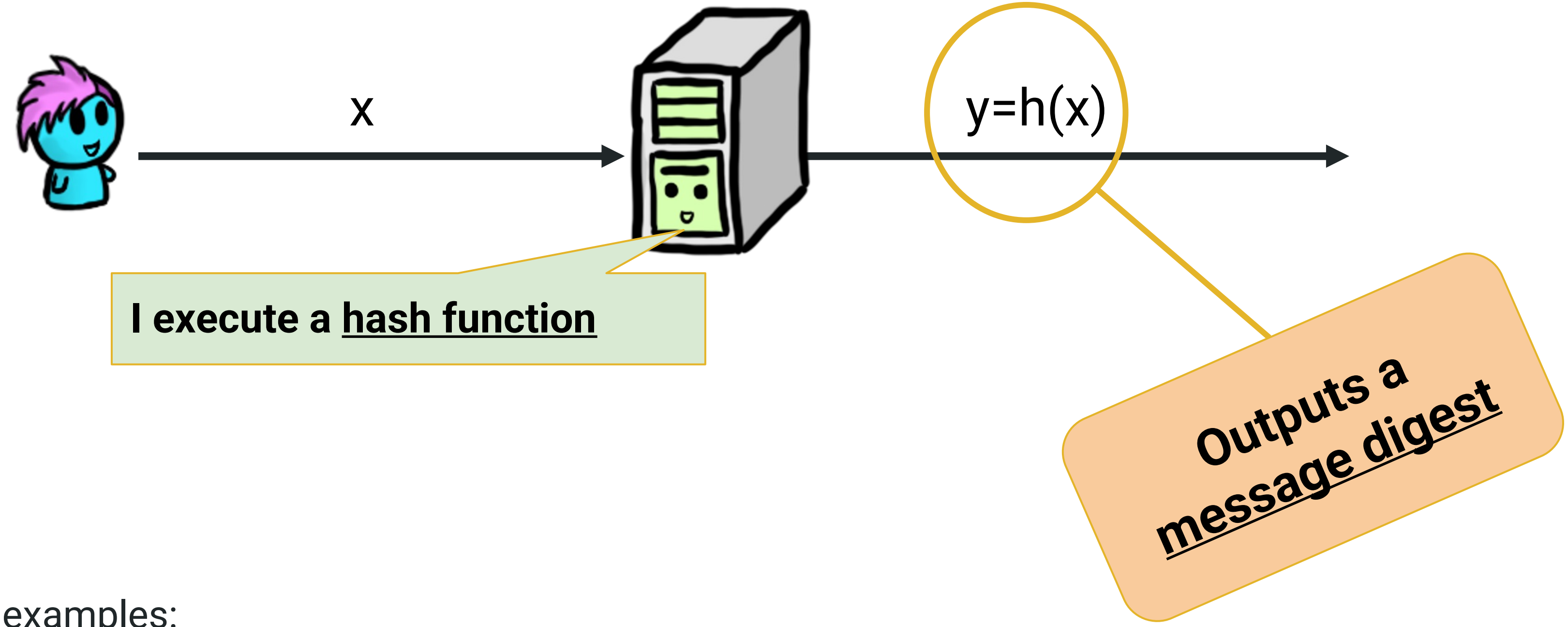



**Takes an arbitrary length string, and computes a fixed length string.**

Common examples:

- MD5, SHA-1, SHA-2, SHA-3 (aka Keccak after 2012)

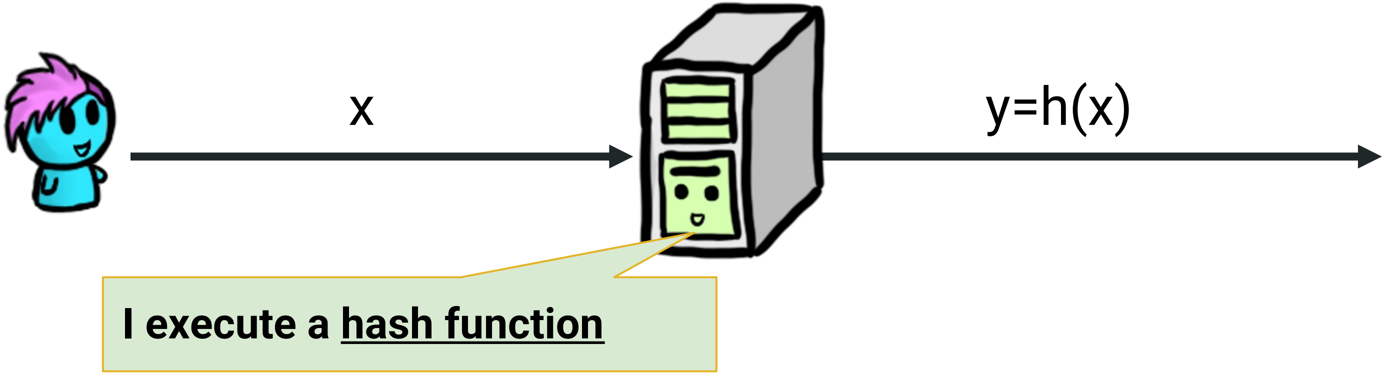
# Towards Integrity: Cryptographic Hash Functions




Common examples:

- MD5, SHA-1, SHA-2, SHA-3 (aka Keccak after 2012)

# Towards Integrity: Cryptographic Hash Functions

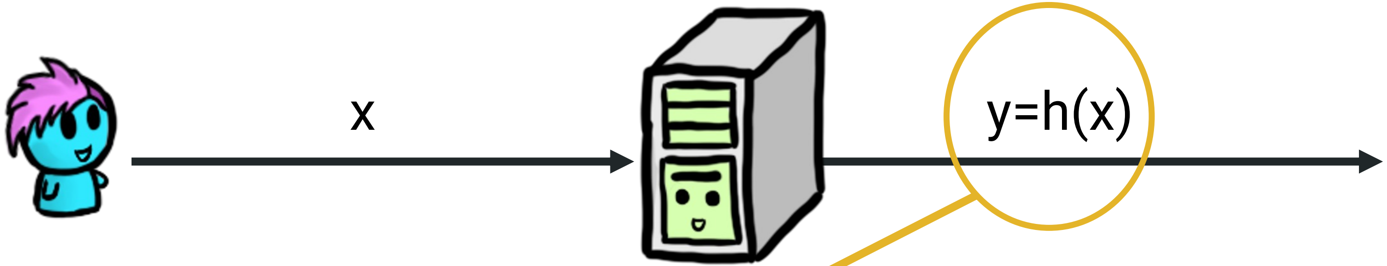



Q: Why is this useful?

Common examples:

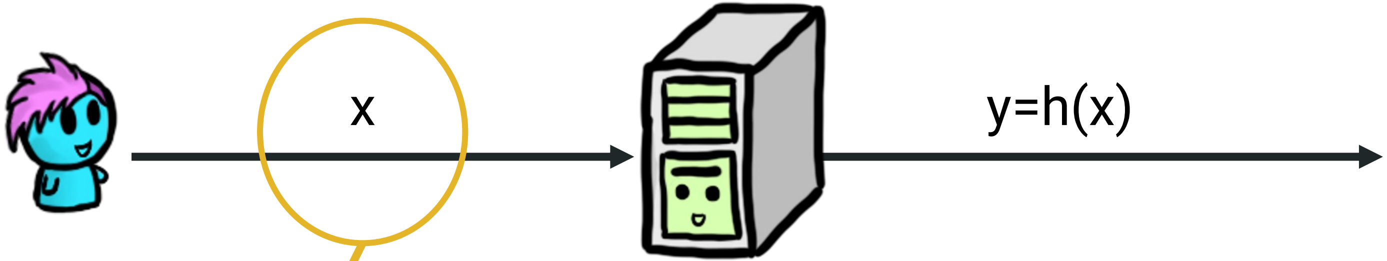
- MD5, SHA-1, SHA-2, SHA-3 (aka Keccak after 2012)

# Properties: Preimage-Resistance



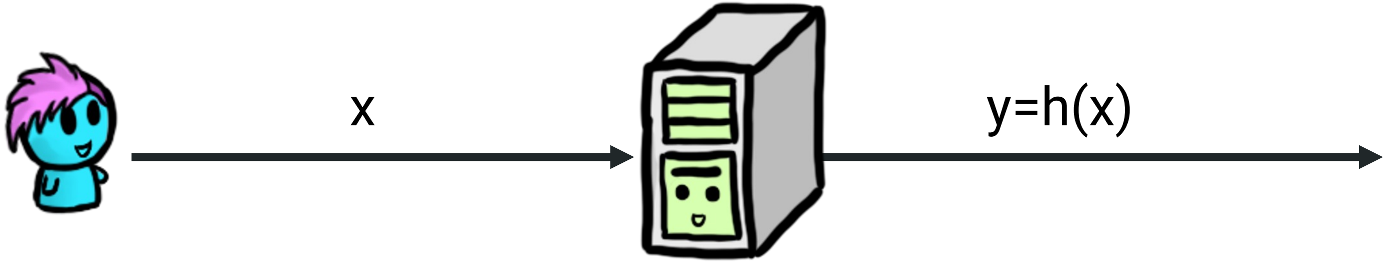

**Goal:** Given y, "hard" to find x such that h(x) = y

# Properties: Second Preimage-Resistance



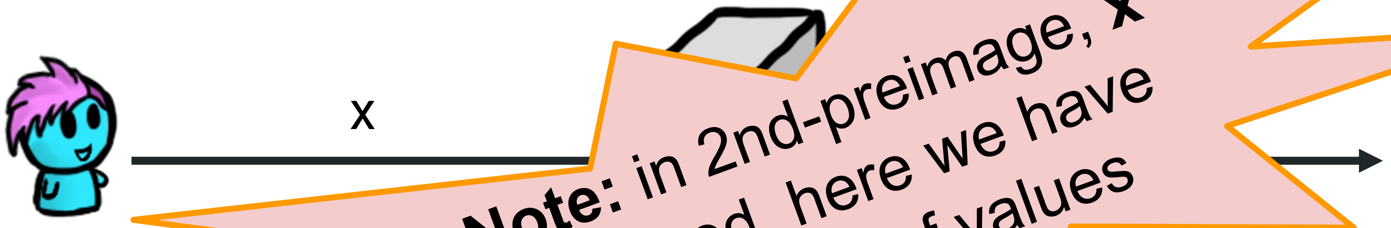

**Goal:** Given x, “hard” to find x’ <> x such that h(x) = h(x’)

# Properties: Collision-Resistance




**Goal:** It’s hard to find any two distinct x, x’ such that h(x) = h(x’)

# Properties: Collision-Resistance




**Note:** in 2nd-preimage, **x** was fixed, here we have free choice of values

**Goal:** It’s hard to find any two distinct x, x’ such that h(x) = h(x’)

# Making it too hard to break these properties?

- SHA-1: takes $2^{160}$ work to find a preimage or second image
- SHA-1: takes $2^{80}$ to find a collision using brute-force search
  - If there are $2^n$ digests, we need to try an average $2^{n/2}$ messages to find 2 with the same digest

# Making it too hard to break these properties?

- SHA-1: takes $2^{160}$ work to find a preimage or second image
- SHA-1: takes $2^{80}$ to find a collision using brute-force search
  - If there are $2^n$ digests, we need to try an average $2^{n/2}$ messages to find 2 with the same digest
- Collisions are always easier to find than preimages or second preimages due to the birthday paradox

# The birthday paradox

- If there are **n** people in a room, what is the probability that at least two people have the same birthday?
- For n = 2: P(2) = 1 – $\frac{364}{365}$
- For n = 3: P(3) = 1 – $\frac{364}{365}$ x $\frac{363}{365}$
- For n people: P(n) = 1 – $\frac{364}{365}$ x $\frac{363}{365}$ x … x $\frac{365-n-1}{365}$

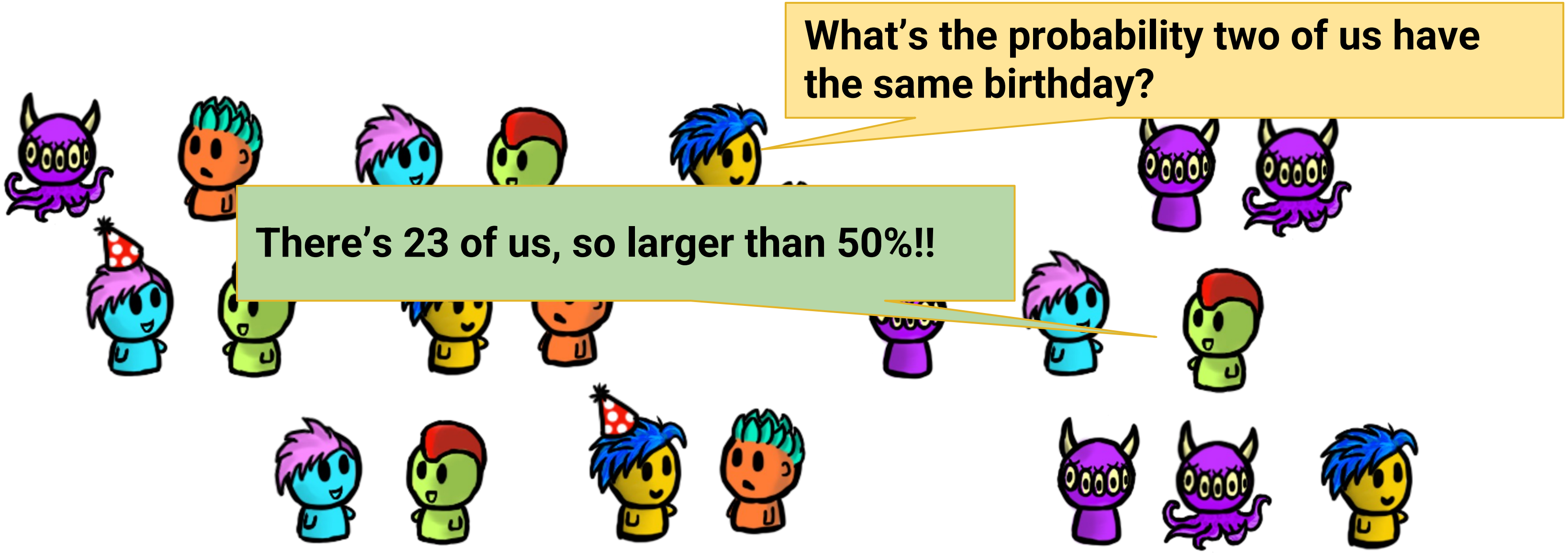
# Collisions and the Birthday Paradox

**Collisions are easier due to the birthday paradox**

**What's the probability two of us have the same birthday?**

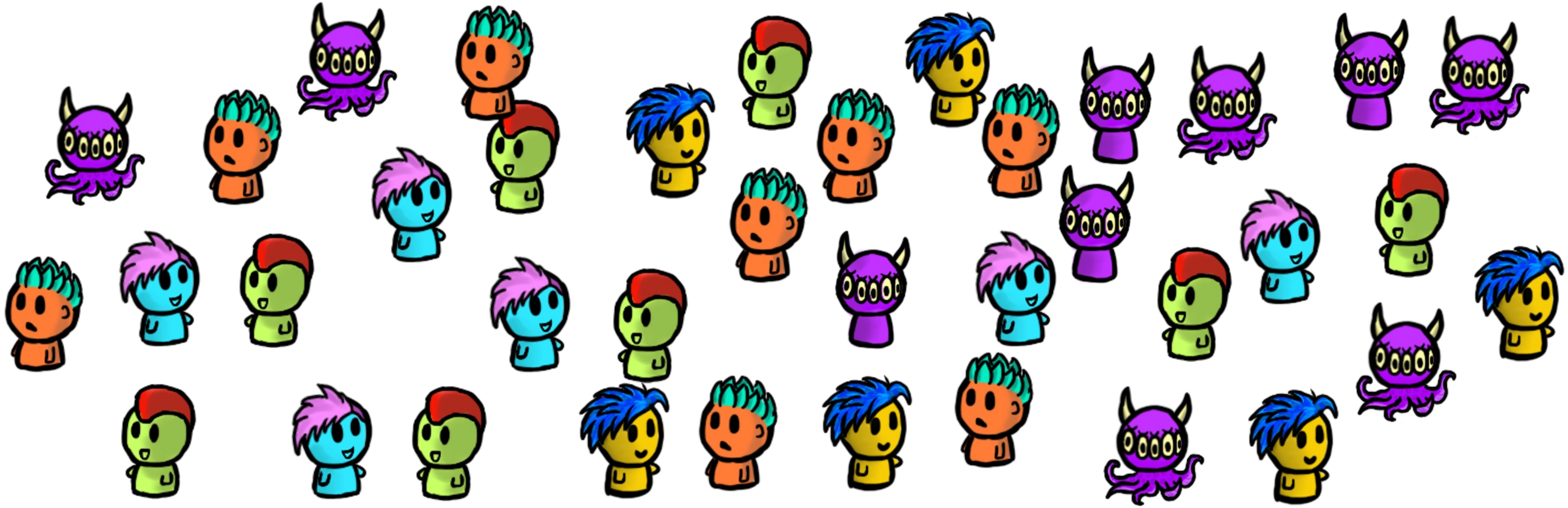
# Collisions and the Birthday Paradox

**Collisions are easier due to the birthday paradox**



# Collisions and the Birthday Paradox

**Collisions are easier due to the birthday paradox**



# Collisions and the Birthday Paradox

**Collisions are easier due to the birthday paradox**

**There’s 40 of us, so almost 90%!!**

# Collisions and the Birthday Paradox

**Collisions are easier due to the birthday paradox**

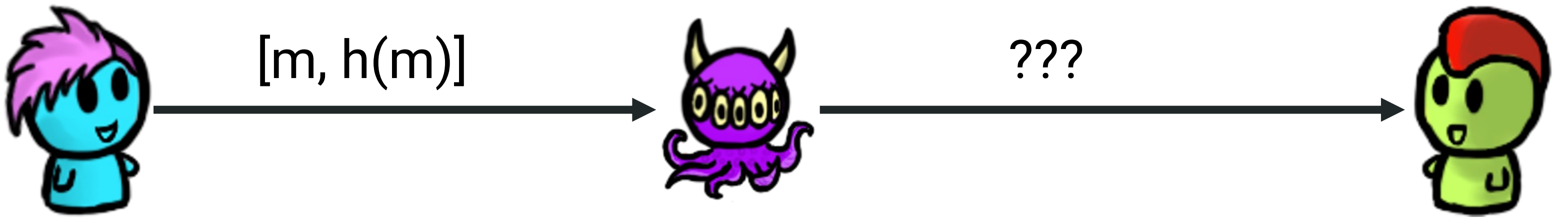
# Collisions and the Birthday Paradox

**Collisions are eas  due t**

Not the end of our problems…

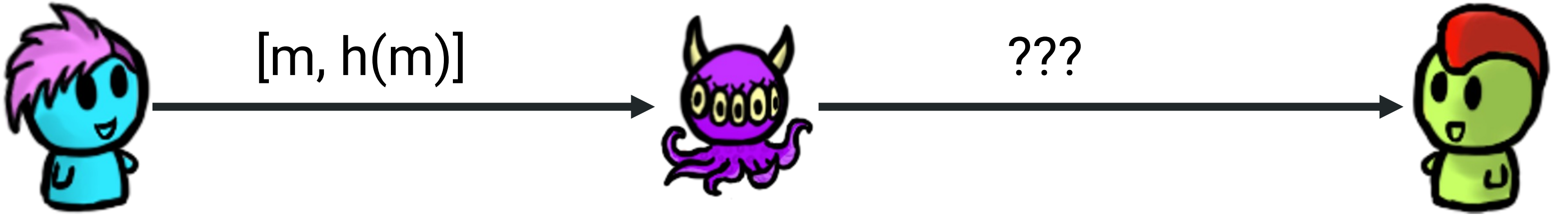
**Th  9%!!!**

# How about a bad example? (Integrity over Conf.)



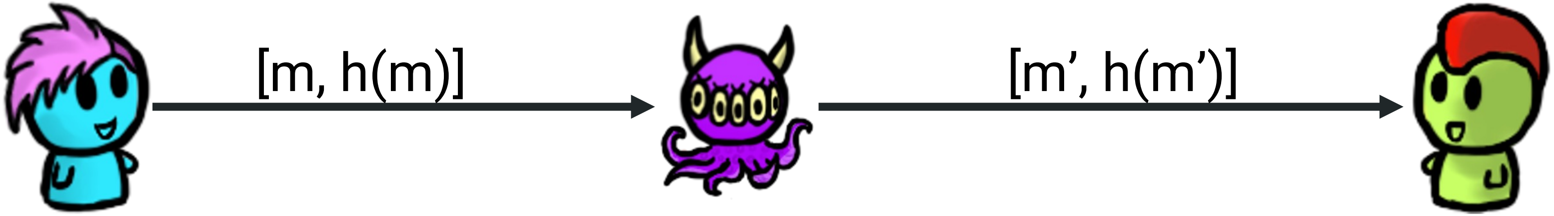

**Q:** What can Mallory do to send the message she wants (change it)?

# How about a bad example? (Integrity over Conf.)




**Q:** What can Mallory do to send the message she wants (change it)?

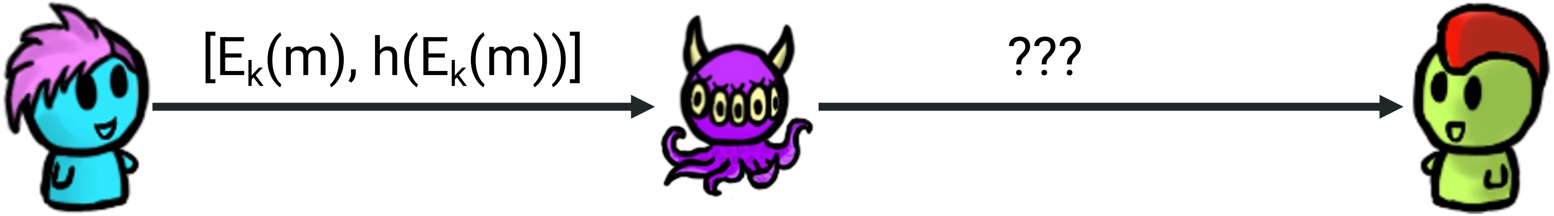
**A:** Just change it…Mallory can compute the new hash herself.



# How about a less bad example? (Integrity & Conf.)

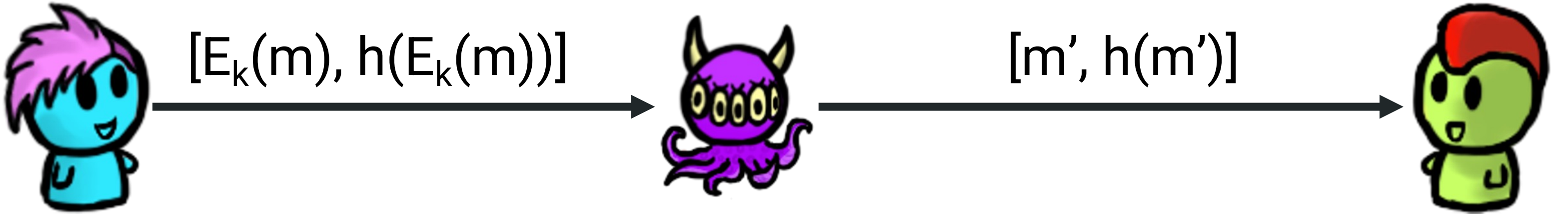

**Q:** What can Mallory do to send the message she wants (change it)?

# How about a less bad example? (Integrity & Conf.)



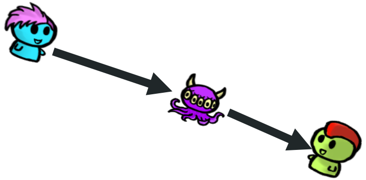

**Q:** What can Mallory do to send the message she wants (change it)?

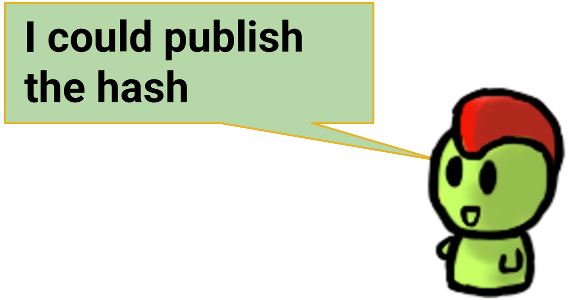
**A:** Still just change it.



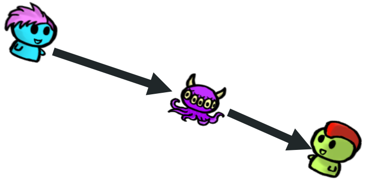
# Limitations for Cryptographic Hash Functions

- Integrity guarantees only when there is a **secure** way of sending/storing the message digest





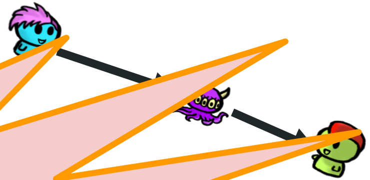
# Limitations for Cryptographic Hash Functions



- Integrity guarantees only when there is a **<u>secure</u>** way of sending/storing the message digest
- E.g.:

# Limitations for Cryptographic Hash Functions

- Integrity guarantees only when there is a **secure** way of sending/storing the message
- E.g.:



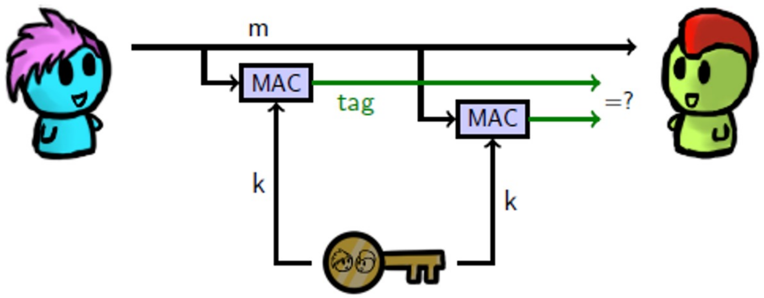
I could publish the hash of my public key on a business card

…would …ace on the card, I …hash to… verify it!

What if…we don't have an external/physical channel?

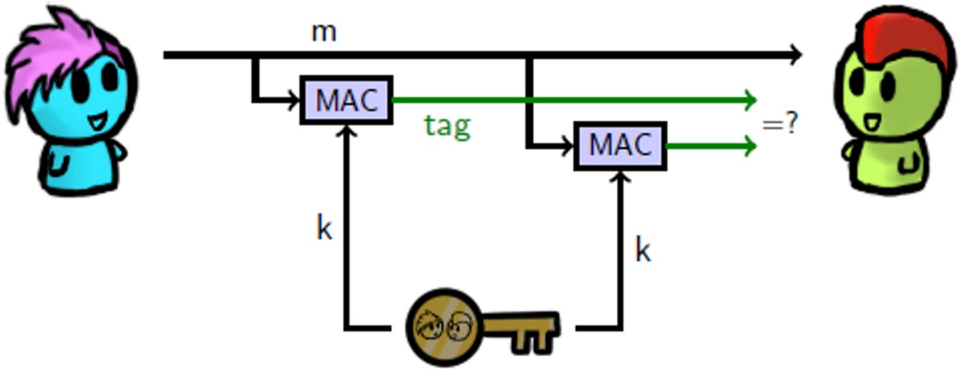
# Authentication and Hash Functions

- Use “keyed hash functions”
- Requires a key to generate or check the hash value (a.k.a., tag)




**Called:** Message authentication codes (MACs)

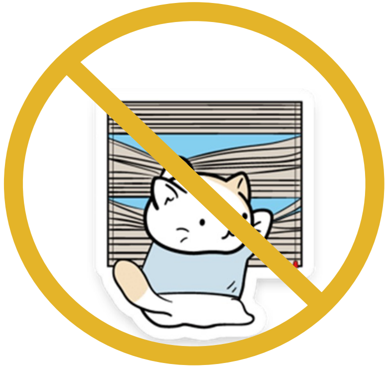
# Message Authentication Codes (MACs)




I don’t have the key to generate or check the values…

Use “keyed hash functions”
e.g., SHA-1-HMAC, SHA-256-HMAC, CBC-MAC

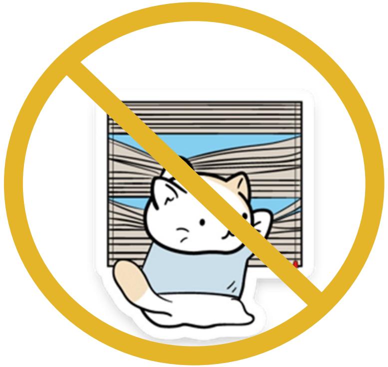
# Combine Ciphers and MACs



**Confidentiality**

**Integrity**

# Combine Ciphers and MACs



**Confidentiality**

**Integrity**

Practical systems need both

# But how to combine them? Three possibilities

- MAC-then-Encrypt
- Encrypt-and-MAC
- Encrypt-then-MAC

# But how to combine them? Three possibilities

- MAC-then-Encrypt
- Encrypt-and-MAC
- E

Ideally, there is an authenticated encryption mode that combines them…but…

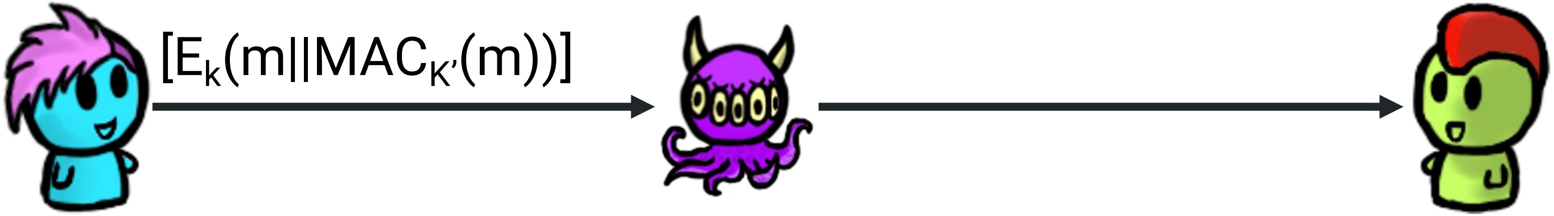
# Let’s make it work?

- Alice and Bob have a secret key **k** for a cryptosystem
- Also, a secret key **K’** for their MAC

How can Alice build a message for Bob in the following three scenarios?

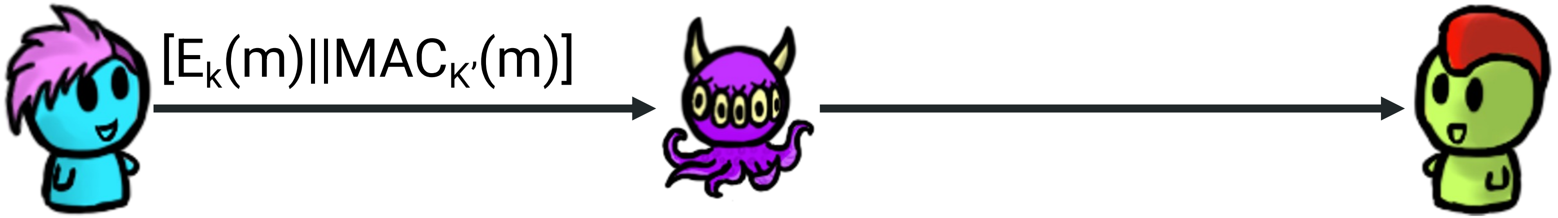
# MAC-then-Encrypt

- Alice and Bob have a secret key **k** for a cryptosystem and a secret key **K’** for their MAC
- Compute the MAC on the message, then encrypt the message and MAC together, and send that ciphertext.



# Encrypt-and-MAC

- Alice and Bob have a secret key **k** for a cryptosystem and a secret key **K’** for their MAC
- Compute the MAC on the message, the encryption of the message, and send both.



# Encrypt-then-MAC

- Alice and Bob have a secret key **k** for a cryptosystem and a secret key **K'** for their MAC
- Encrypt the message, compute the MAC on the encryption, send encrypted message and MAC

# Which order is correct?

**Q:** Which should be recommended then?

$E_k(m||MAC_{K'}(m))$ **vs.** $E_k(m)||MAC_{K'}(m)$ **vs.** $E_k(m)||MAC_{K'}(E_k(m))$

**MAC-then-encrypt** **Encrypt-and-MAC** **Encrypt-then-MAC**

# The Doom Principle


"if you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will somehow inevitably lead to doom."

# The Doom Principle


"if you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will somehow inevitably lead to doom."

**Q:** What are possible problems that can arise from the orderings?

# The Doom Principle


**Q:** What are possible problems that can arise from the orderings?

- **MAC-then-Encrypt:** Allows an adversary to force Bob into decrypting the ciphertext before verifying the MAC. May lead to a padding oracle attack
- **Encrypt-and-MAC:** Allows an adversary to force Bob into decrypting the ciphertext to verify the MAC. May lead to a chosen-ciphertext attack

# The Doom of MAC-then-Encrypt


**Observation:** To verify the MAC, Bob has first to decrypt the message, since the MAC is part of the encrypted payload

- **Padding oracle attack:** The idea is for the attacker to send modified ciphertexts to Bob and observe how he responds.

- With CBC, by modifying the last block of the ciphertext in a way that alters the block's padding, the attacker can tell if the padding is valid or not.

- If the padding is invalid, the system might respond differently (e.g., with an error message that is padding-specific). This information leakage allows the attacker to gradually decrypt the ciphertext byte by byte.

# The Doom of Encrypt-and-MAC


**Q:** What happens if the MAC has no mechanism to provide confidentiality?

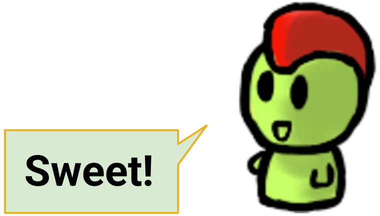
- MACs are meant to provide integrity
- MACs are often implemented by a **deterministic** algorithm without an explicit random input (essentially, for a given key and message, the output of the MAC is always the same).
- If a deterministic MAC is used, then there is no guarantee that the tag $E_k(m)||\mathbf{MAC_{K'}(m)}$ will not leak information about the secret message **m**.

# Which order is correct?

**Usually:** we want the receiver to verify the MAC first!

**Recommended:** Encrypt-then-MAC, $E_k(m)||MAC_{K'}(E_k(m))$

- **Encrypt-then-MAC:** Allows Bob to check the MAC of the ciphertext before performing any decryption whatsoever (e.g., prevent attacks by immediately closing a connection if the MAC fails)



# More properties that matter?

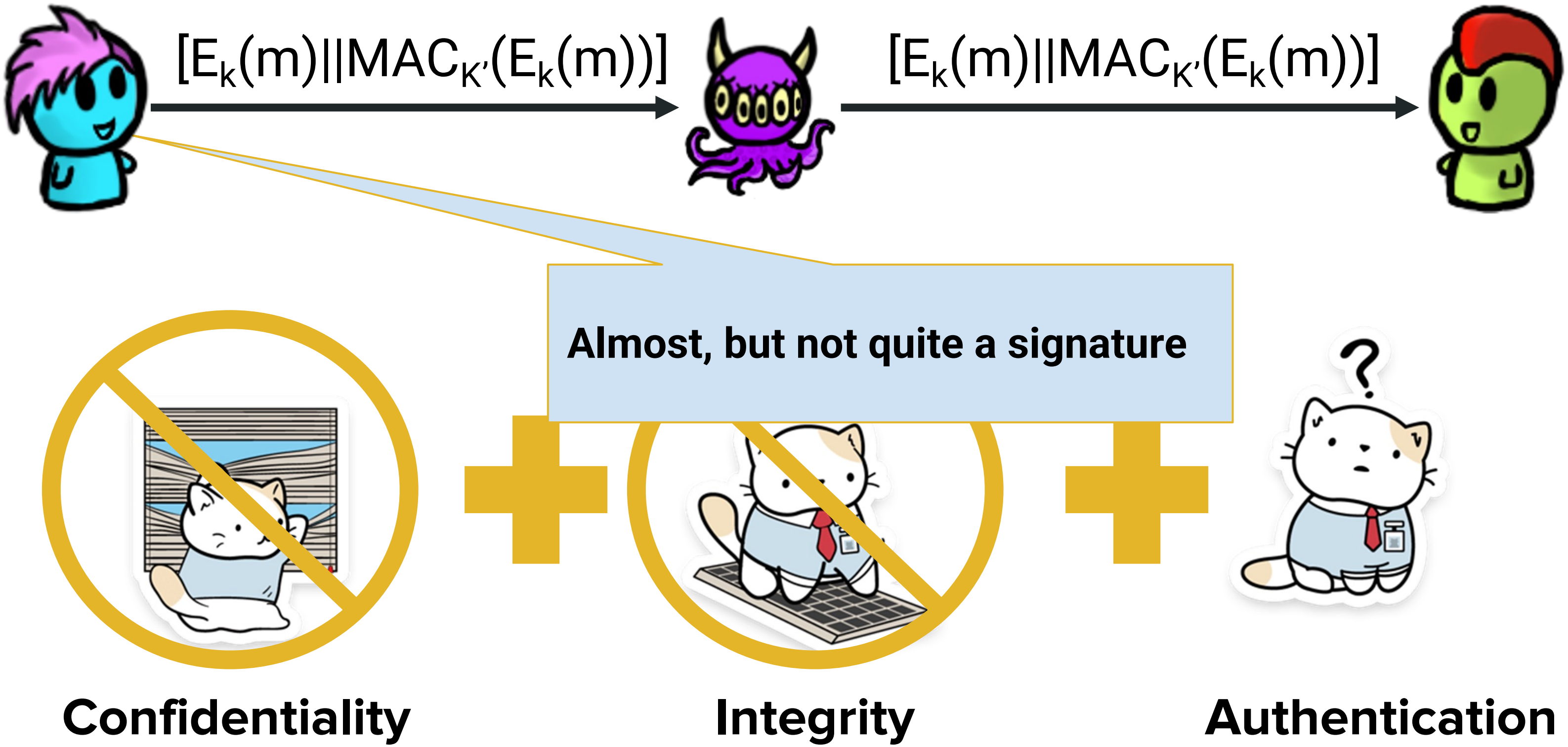
# Repudiation


**Alice sent *m*, and I received the same m she sent.**

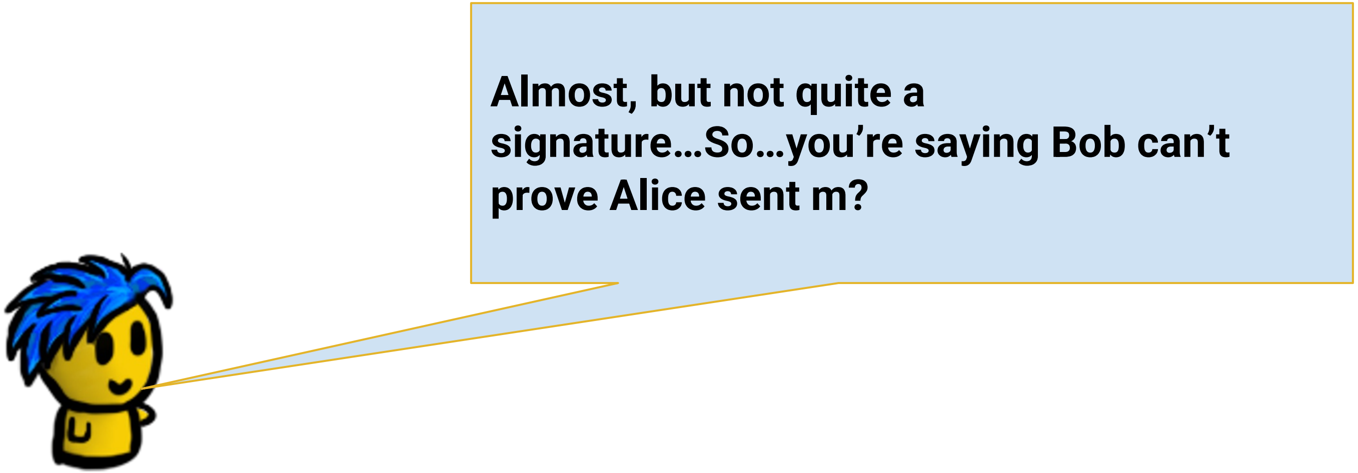
# Repudiation


**Confidentiality** **Integrity** **Authentication**

# Repudiation



# Repudiation

# Repudiation


**Almost, but not quite a signature…So…you're saying Bob can't prove Alice sent m?**

**Q:** Why can't Bob prove it?

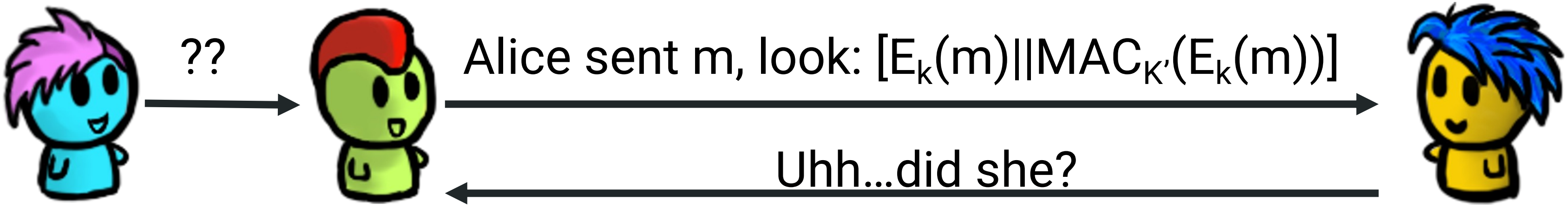
# Repudiation


**Almost, but not quite a signature…So…you're saying Bob can't prove Alice sent m?**

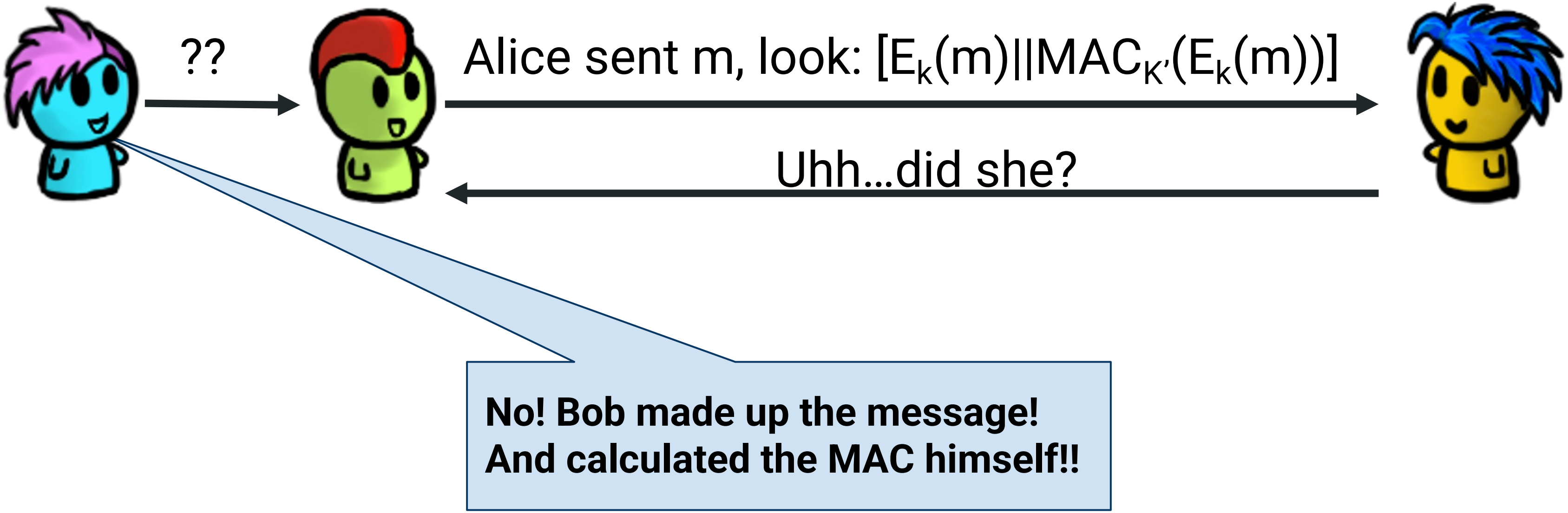
**Q:** Why can't Bob prove it?

**A:** Either Alice or Bob could create any message and MAC combo…also Carol doesn't know the secret keys.

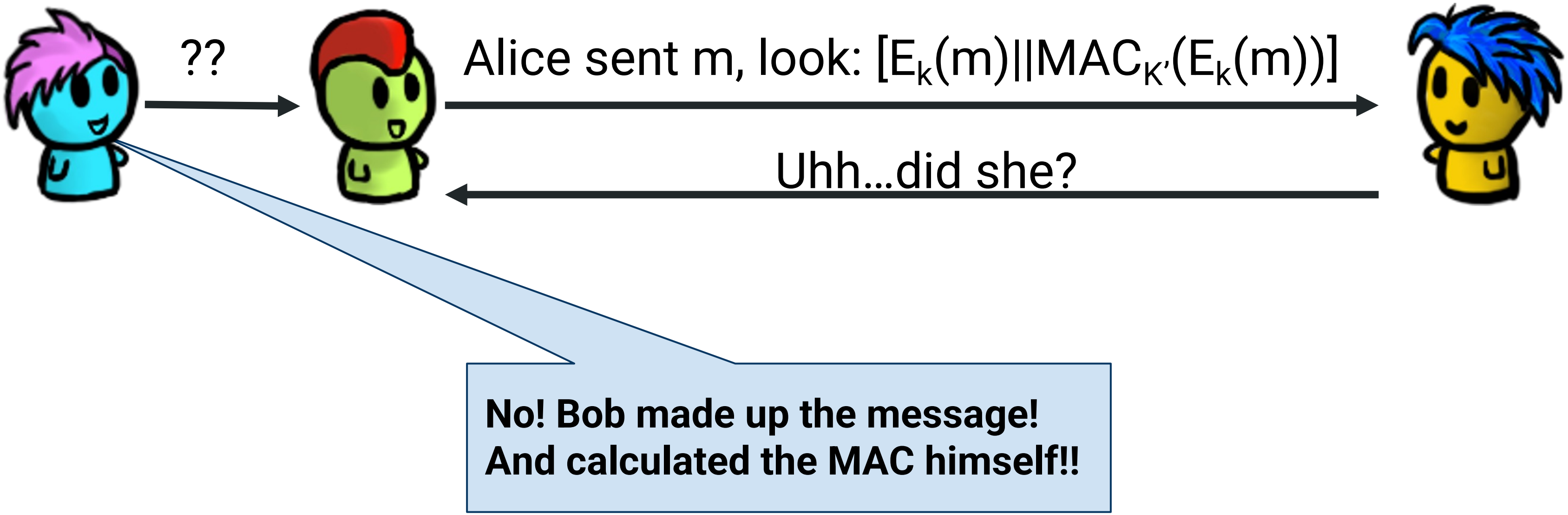
# Implications? Repudiation Con't



# Implications? Repudiation Con't

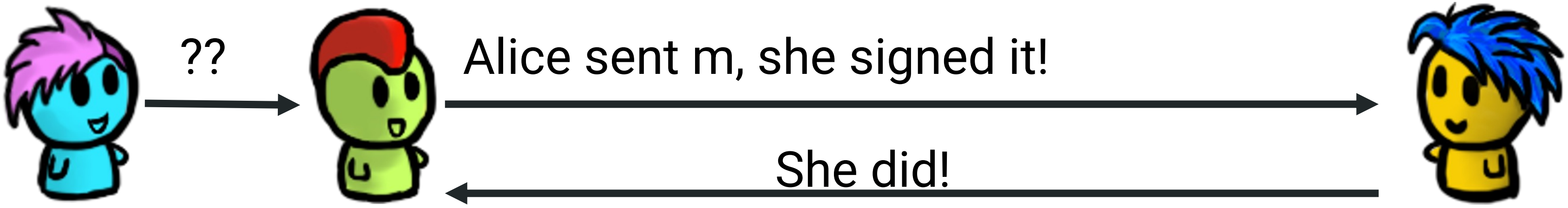


# Implications? Repudiation Con’t

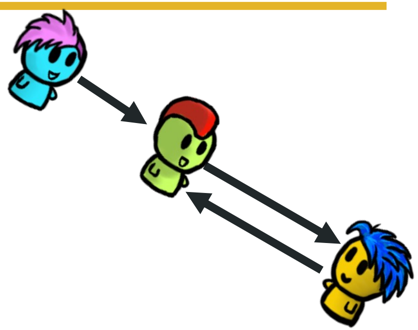



**Repudiation Property:** For some applications this property is good (e.g., private conversations)…others less good (e.g., e-commerce…).

# Digital Signatures - For When Repudiation is Bad

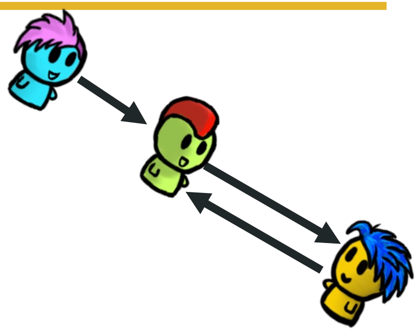


# Properties and Goals from Digital Signatures



If Bob receives a message with Alice's digital signature then it should mean:

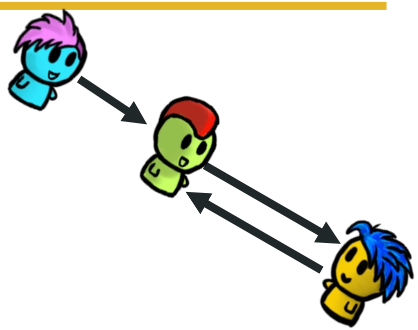
# Properties and Goals from Digital Signatures



If Bob receives a message with Alice's digital signature then it should mean:

- Alice sent it (not  ), like a MAC

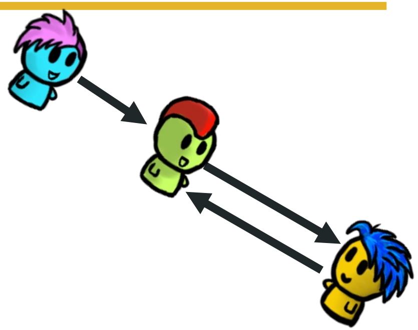
# Properties and Goals from Digital Signatures



If Bob receives a message with Alice's digital signature then it should mean:

- Alice sent it (not  ), like a MAC
- The message was not altered after sending, like a MAC

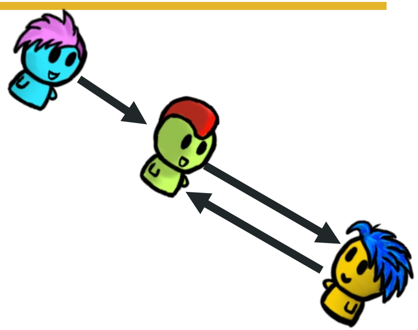
# Properties and Goals from Digital Signatures



If Bob receives a message with Alice’s digital signature then it should mean:

- Alice sent it (not  ), like a MAC
- The message was not altered after sending, like a MAC
- The above two properties should be **provable** to a third party, not like a MAC

# Properties and Goals from Digital Signatures



If Bob receives a message with Alice's digital signature then it should mean:

- Alice sent it (not  ), like a MAC
- The message was not altered after sending, like a MAC
- The above two properties should be **provable** to a third party, not like a MAC

**Achievable?** Use techniques similar to public-key crypto (last class)

# Making Digital Signatures

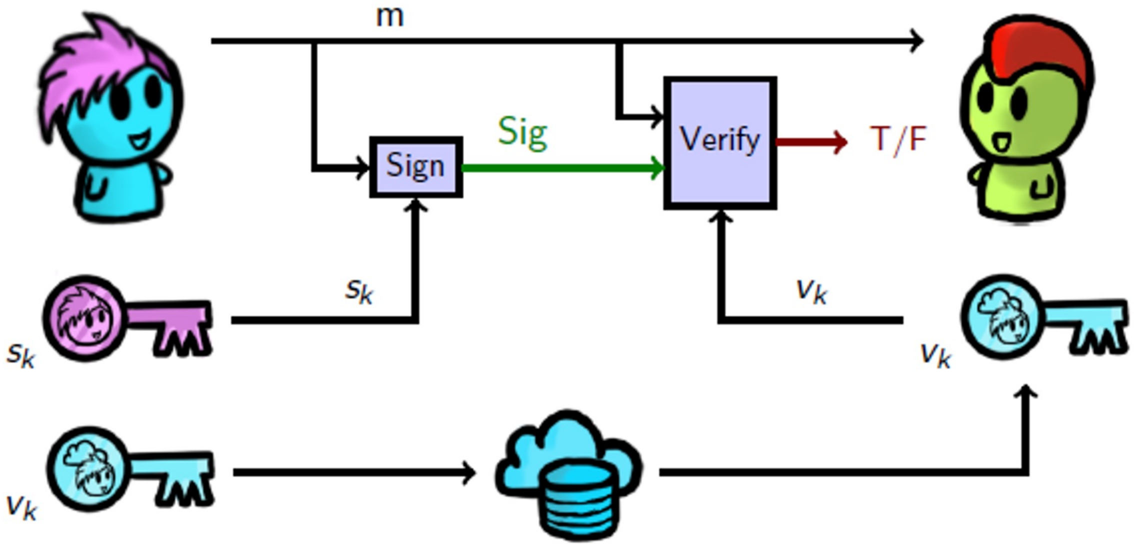
1. A pair of keys

2. Everyone gets each other public verification key

3. Alice signs with private signing key

4. Bob verifies using Alice's public verification key
5. If it verifies correctly, success, valid signature

# Digital Signatures at a Glance

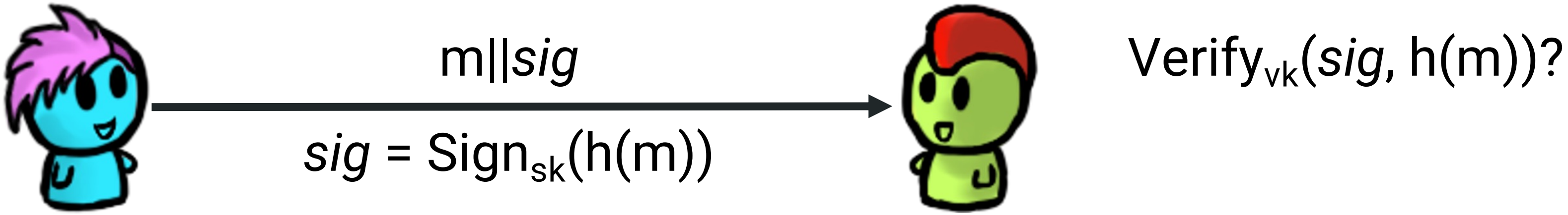


# Faster Signatures, aka More Hybrids

- Signing large messages, slow
- However, a hash is much smaller than the message…

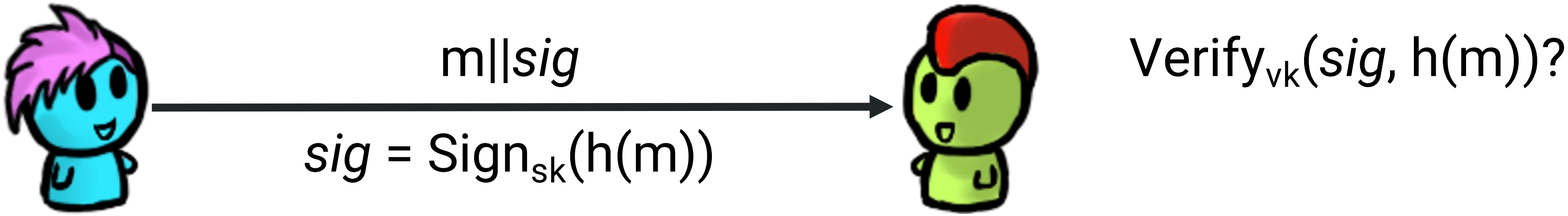
# Faster Signatures, aka More Hybrids

- Signing large messages, slow
- However, a hash is much smaller than the message…



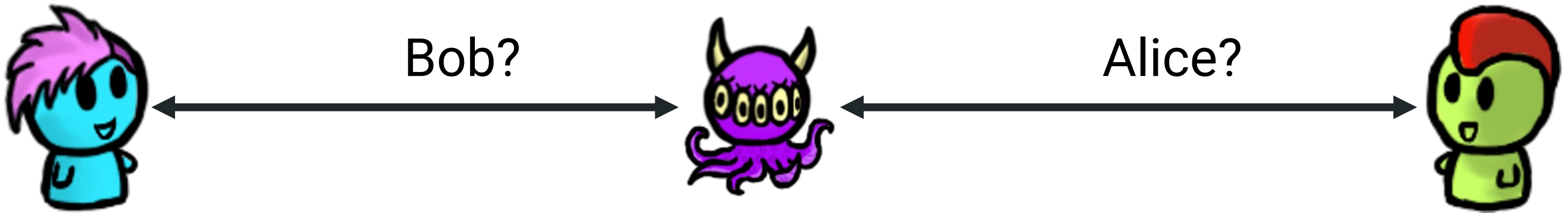
# Faster Signatures, aka More Hybrids

- Signing large messages, slow
- However, a hash is much smaller than the message…



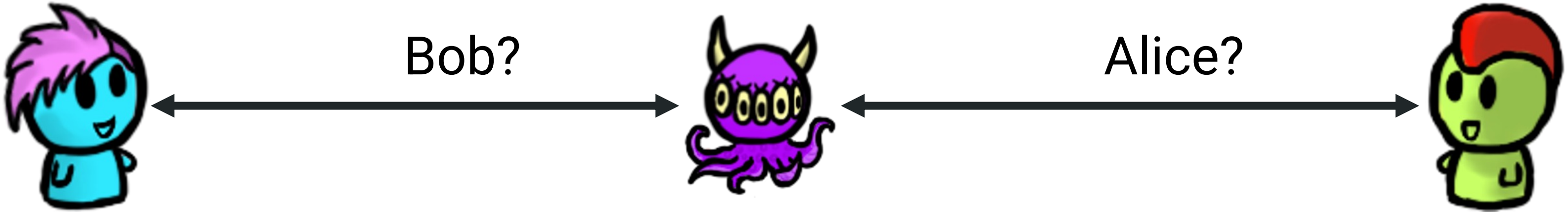

- Finally, authenticity and confidentiality are separate, you need to include both if you want to achieve both

# The Key Management Problem




**Q:** How can Alice and Bob be sure they're talking to each other?

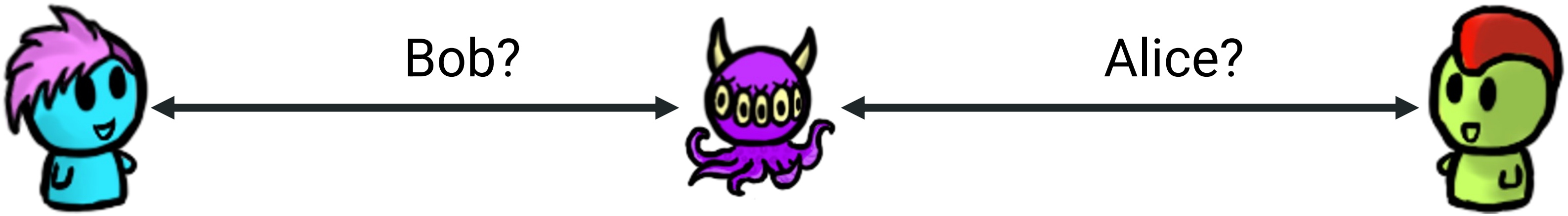
# The Key Management Problem




**Q:** How can Alice and Bob be sure they're talking to each other?

**A:** By having each other's verification key!

# The Key Management Problem

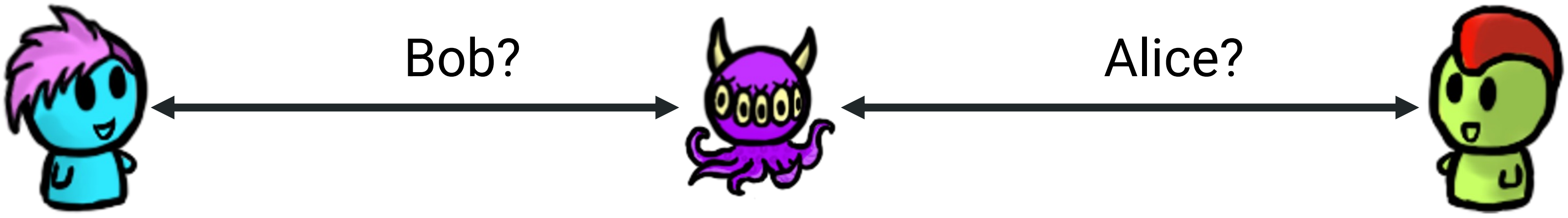



**Q:** How can Alice and Bob be sure they're talking to each other?

**A:** By having each other's verification key!

**Q:** But how do they get the keys…

# The Key Management Problem…Solutions?




**Q:** But how do they get the keys…

**A:** Know it personally (manual keying e.g., SSH)

**A:** Trust a friend (web of trust e.g, PGP)

**A:** Trust some third party to tell them (CAs, e.g., TLS/SSL)

# Nex up: More Cryptography…