

CS459/698

Privacy, Cryptography, Network and Data Security

Integrity and Authenticated Encryption

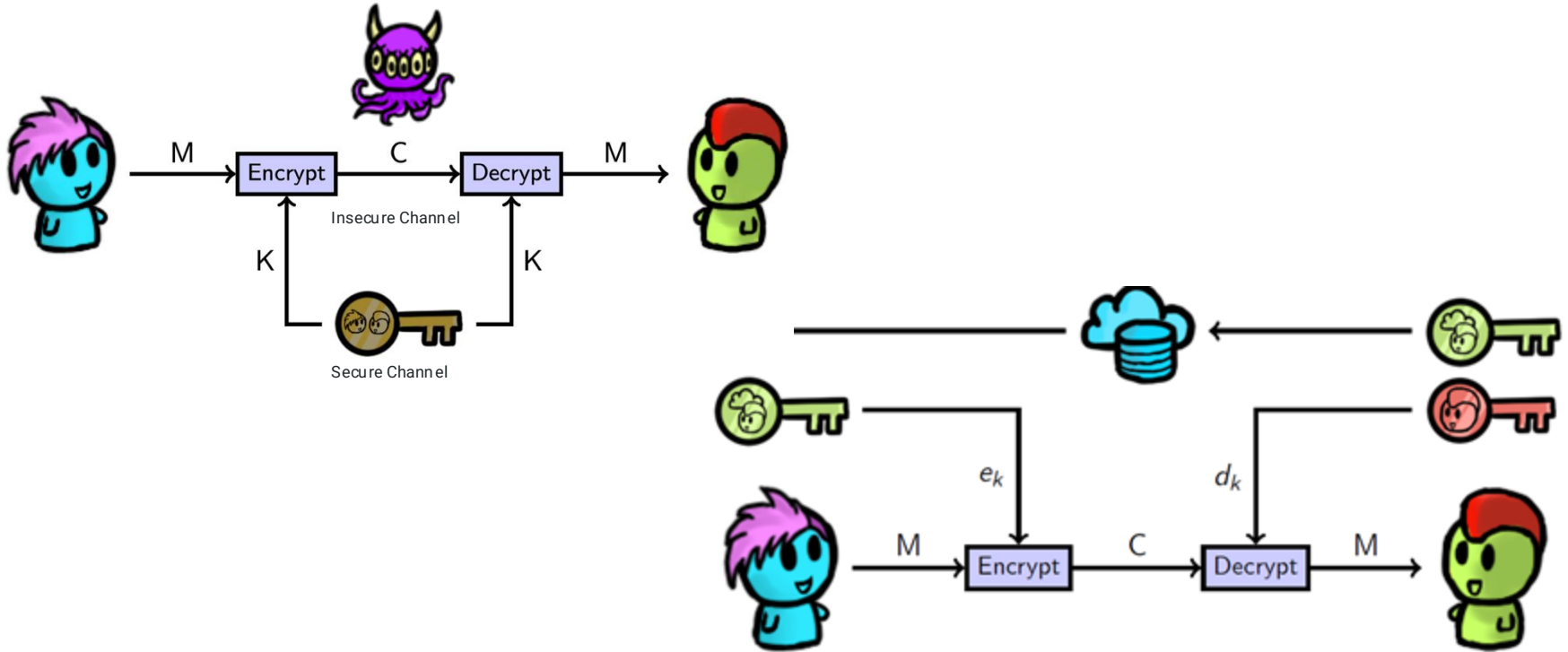
Spring 2025, Monday/Wednesday 02:30pm-03:50pm

Hello

- My name is Abdelkarim Kati
 - You can call me Karim
- I am a Postdoctoral researcher
@ CrySP Lab



Block/Stream Ciphers, Public Key Cryptography...



Is that all there is?



**Modify all messages.
Muhahahah.**

Goal: How do we make sure that Bob gets the same message Alice sent?

Symmetric

Ciphers

Hash
Functions

Message
Auth. codes

PRFs

Stream

Block

Asymmetric

PKE

Digital
Signatures

Key
Exchange

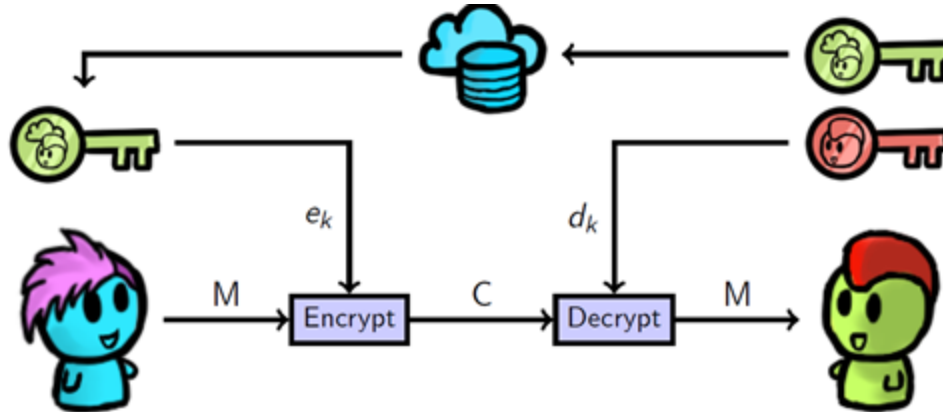
RSA

IND-CCA security types



Integrity components

How do we tell if a message has changed in transit?

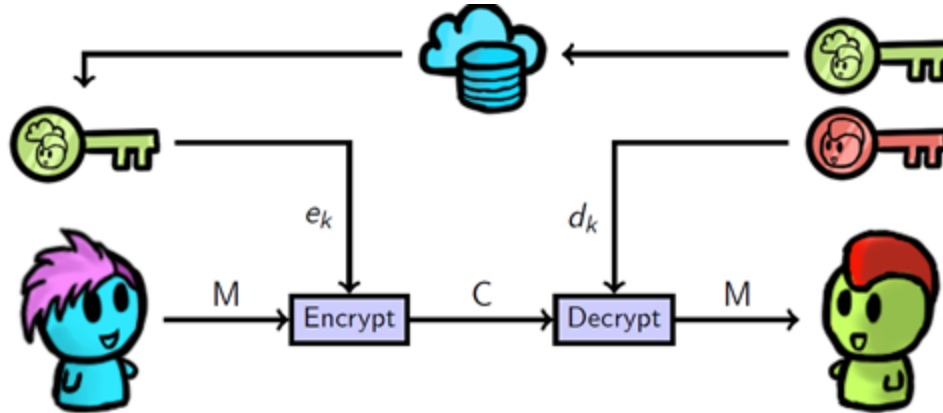


...wait...is this the message Alice sent?



Integrity components

How do we tell if a message has changed in transit?



...wait...is this the message Alice sent?

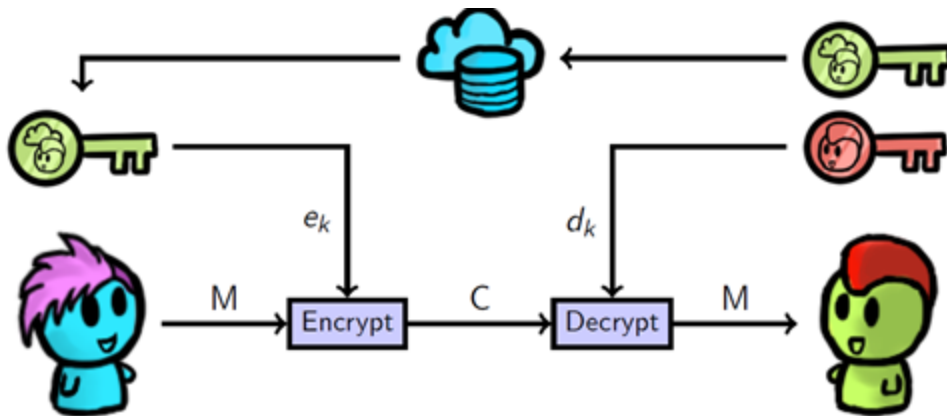
Checksums





Integrity components

How do we tell if a message has changed in transit?



Checksums



Add up all the bytes of M, append the checksum to M so Bob can verify it

Not. Good. Enough.



**Checksums are
deterministic...**

Mallory can easily change the message in such a way that the checksum stays the same.

Message 1: "Hello World!" → Checksum: 61

Message 2: "World Hello!" → Checksum: 61

Message 1': "Hi" → Checksum: 177

Message 2': "DM" → Checksum: 177

Not. Good. Enough.

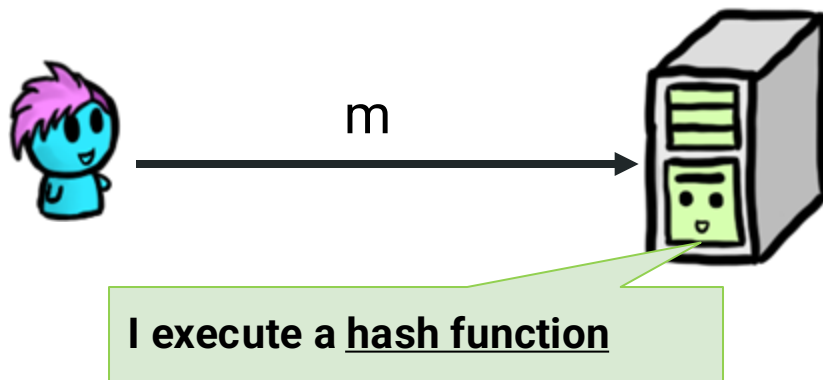


Checksums are deterministic... **I can construct fake ones.**

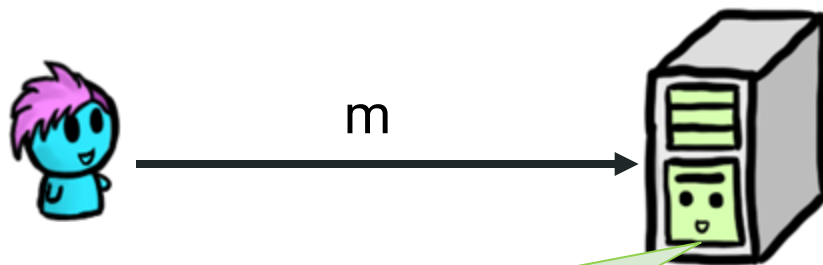
Goal: Make it harder for Mallory to find a second message with the same checksum as the “**real**” message

“**Cryptographic**” checksum

Cryptographic hash functions



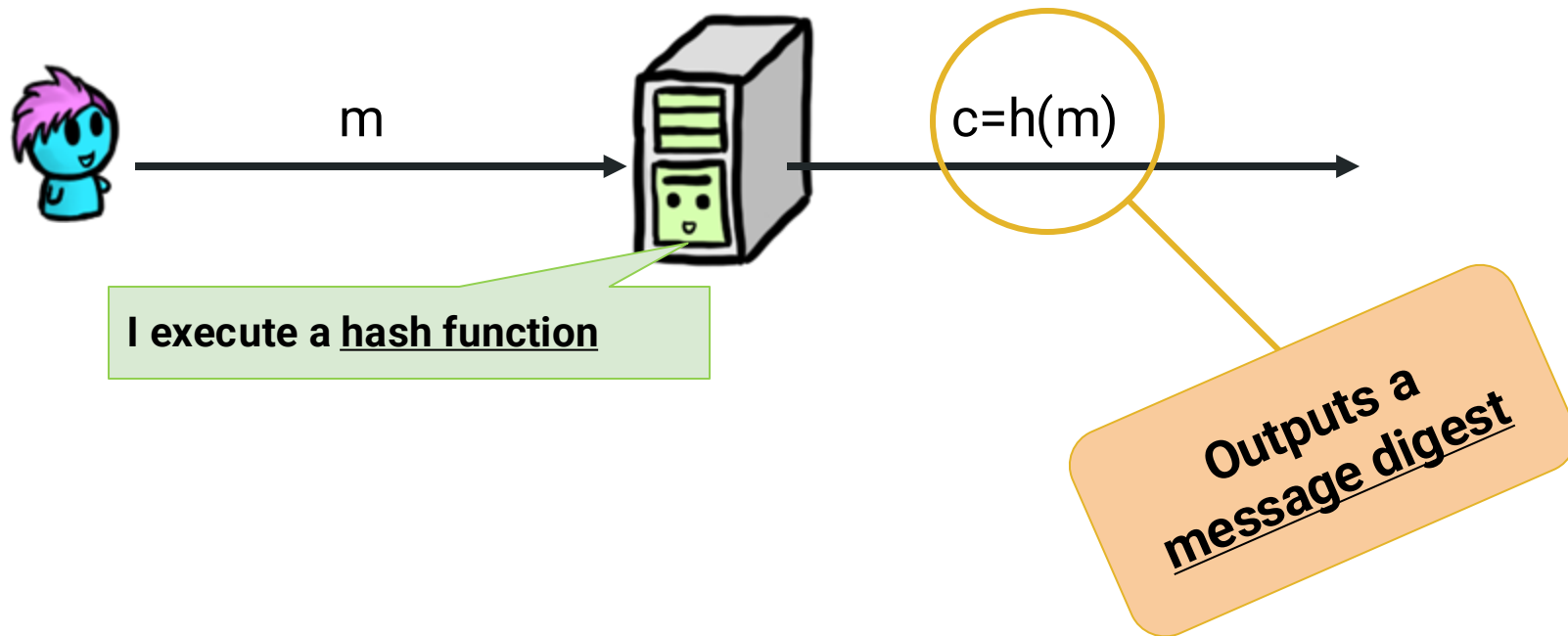
Cryptographic hash functions



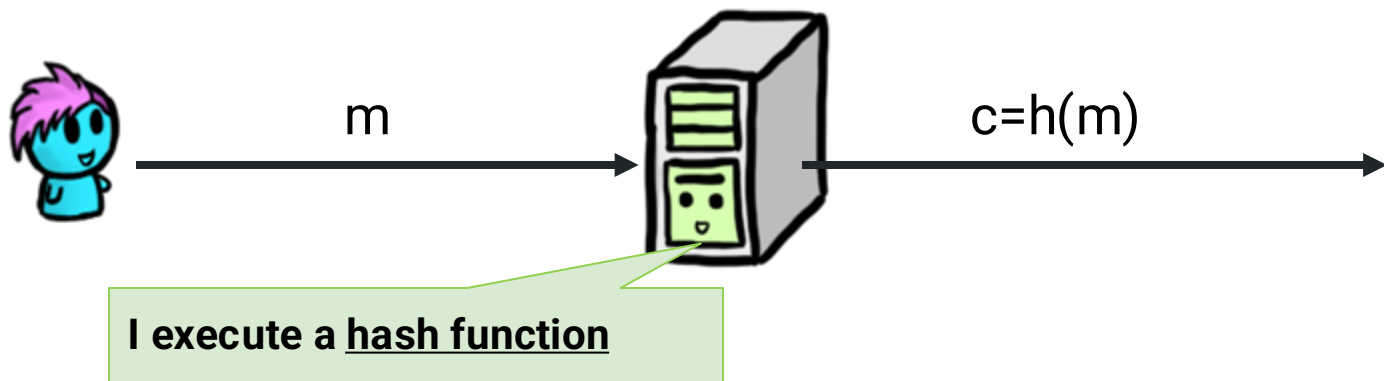
I execute a hash function

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

Cryptographic hash functions



Cryptographic hash functions

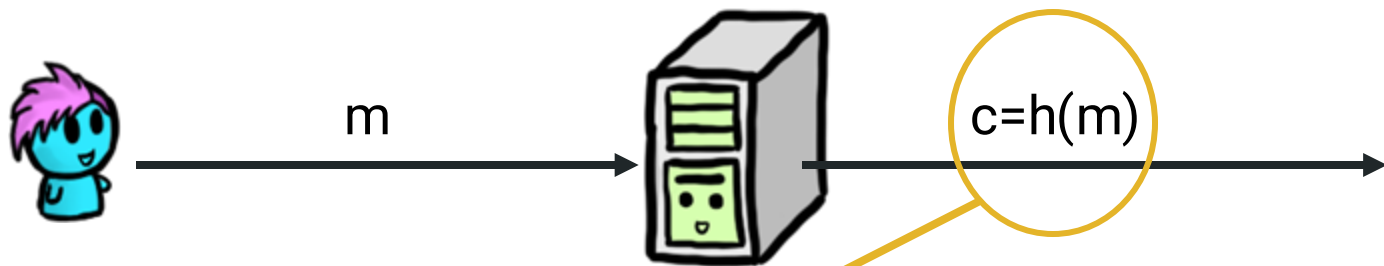


Q: Why is this useful?

Common examples:

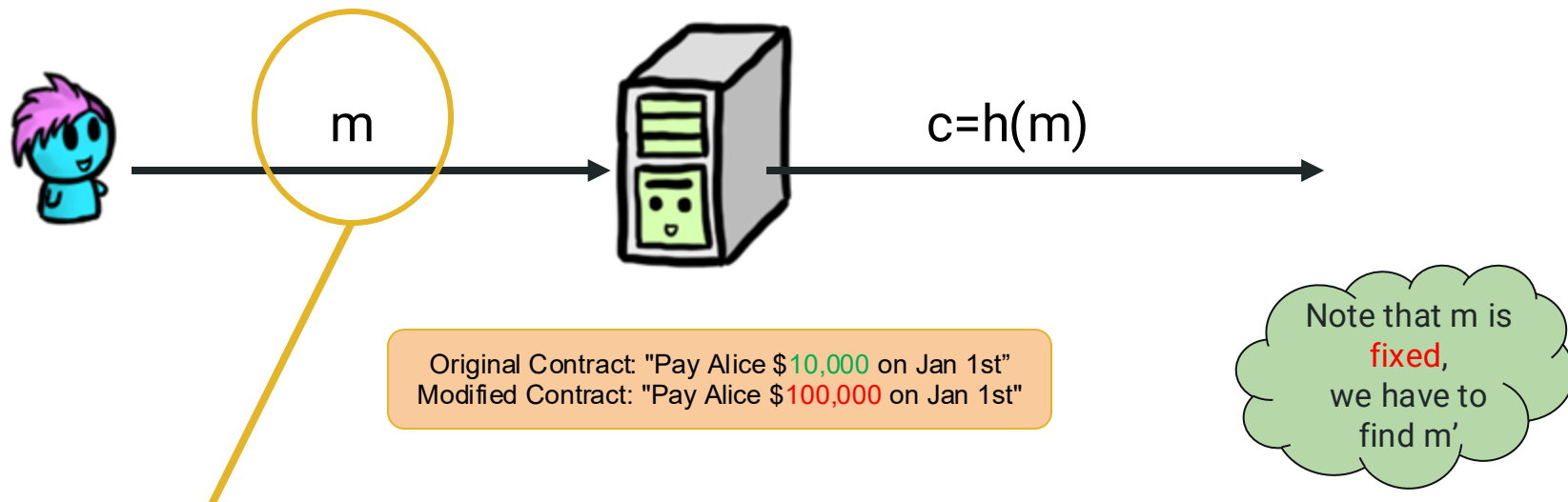
- MD5, SHA-1, SHA-2, SHA-3 (aka Keccak after 2012)

Properties: Preimage-Resistance



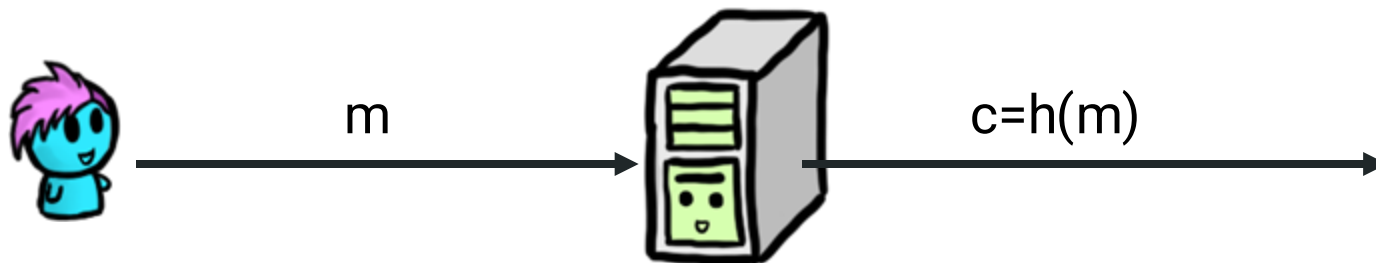
Goal: Given c , it's “hard” to find m such that $h(m) = c$
(i.e., a “preimage” of x)

Properties: Second Preimage-Resistance



Goal: Given m , it's "**hard**" to find $m' \neq m$ such that $h(m) = h(m')$
(i.e., a "**second preimage**" of $h(x)$)

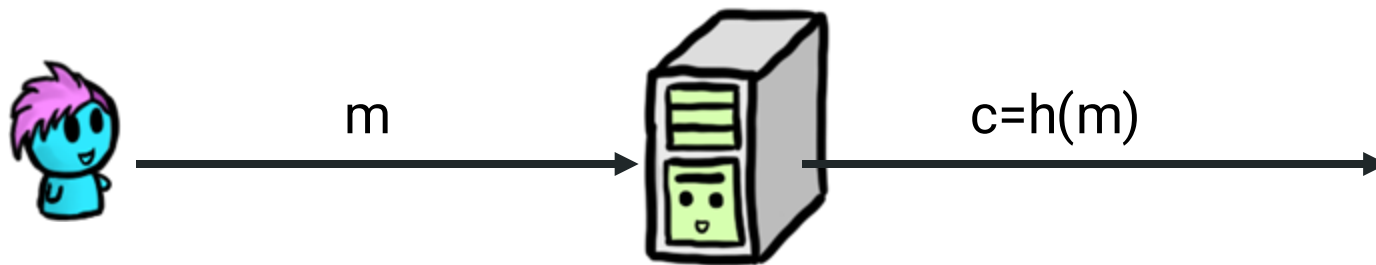
Properties: Collision-Resistance



Note that we have
free choice of
 m and m'

Goal: It's hard to find any two distinct m, m' such that $h(m) = h(m')$
(i.e., a "collision")

Cryptographic hash functions



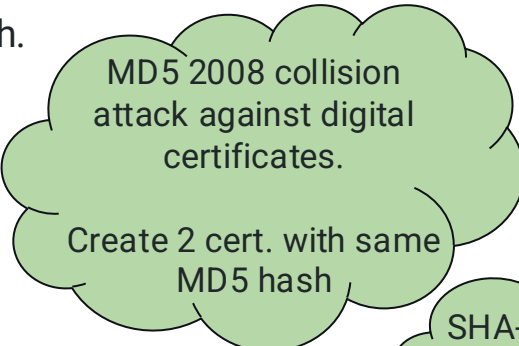
Cryptographic hash functions are designed to resist this sort of thing.
→ They don't always succeed.

What do we mean by “hard”?

- SHA-1: takes 2^{160} work to find a preimage or second image
 - Must find specific input that produces a given hash using brute-force
- SHA-1: takes 2^{80} to find a collision using birthday attack
 - For a hash function with an n -bit output, the birthday attack can find collisions in approximately $2^{n/2}$ (2^{80}) computations.
 - Not looking for a specific match, just ANY match.
 - Collision are easier to find (birthday paradox)

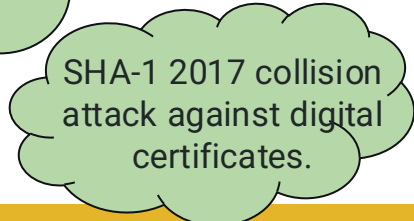
What do we mean by “hard”?

- SHA-1: takes 2^{160} work to find a preimage or second image
 - Must find specific input that produces a given hash using brute-force
- SHA-1: takes 2^{80} to find a collision using birthday attack
 - For a hash function with an n -bit output, the birthday attack can find collisions in approximately $2^{n/2}$ (2^{80}) computations.
 - Not looking for a specific match, just ANY match.
 - Collision are easier to find (birthday paradox)



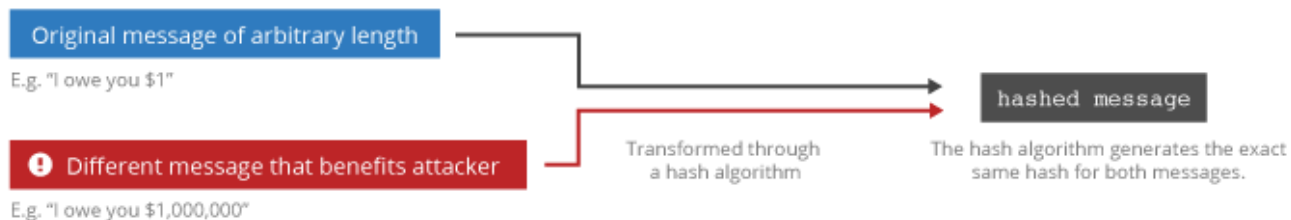
MD5 2008 collision
attack against digital
certificates.

Create 2 cert. with same
MD5 hash

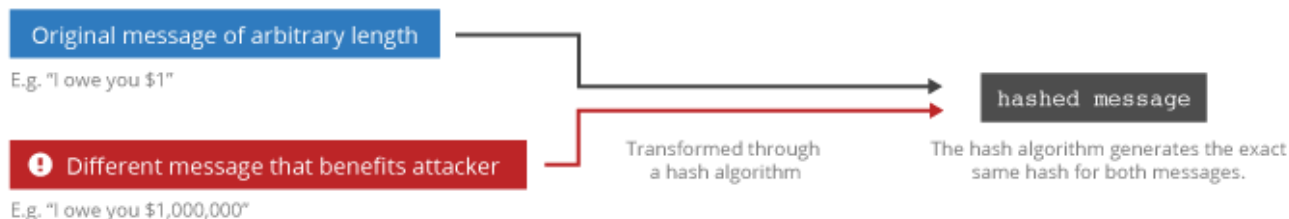


SHA-1 2017 collision
attack against digital
certificates.

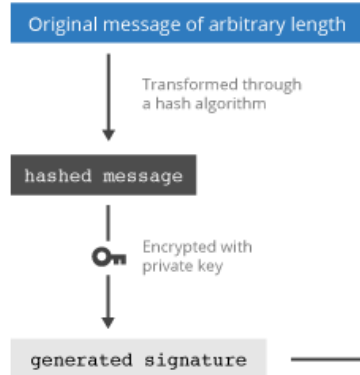
How collisions work



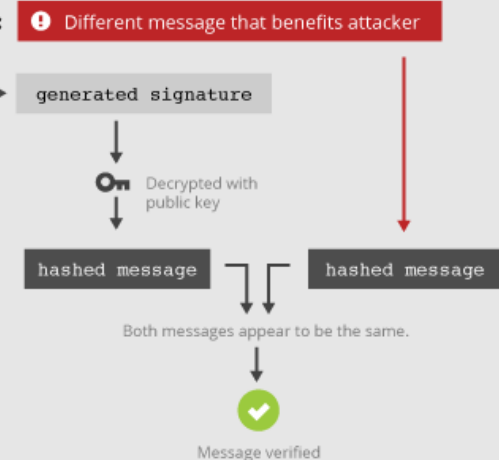
How attackers exploit hash collisions



A normal message is written and signed



The message is altered before it can be verified



The birthday paradox

If there are n people in a room, what is the probability that at least two people have the same birthday?

$$P(\text{collision}) = 1 - \prod_{i=0}^{n-1} \frac{d-i}{d}.$$

- For $n = 2 \rightarrow P(\text{collision}) = 1 - \frac{364}{365}$
- For $n = 3 \rightarrow P(\text{collision}) = 1 - \frac{364}{365} \times \frac{363}{365}$
- For n people $\rightarrow P(\text{collision}) = 1 - \frac{364}{365} \times \frac{363}{365} \times \dots \times \frac{365-n+1}{365}$

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

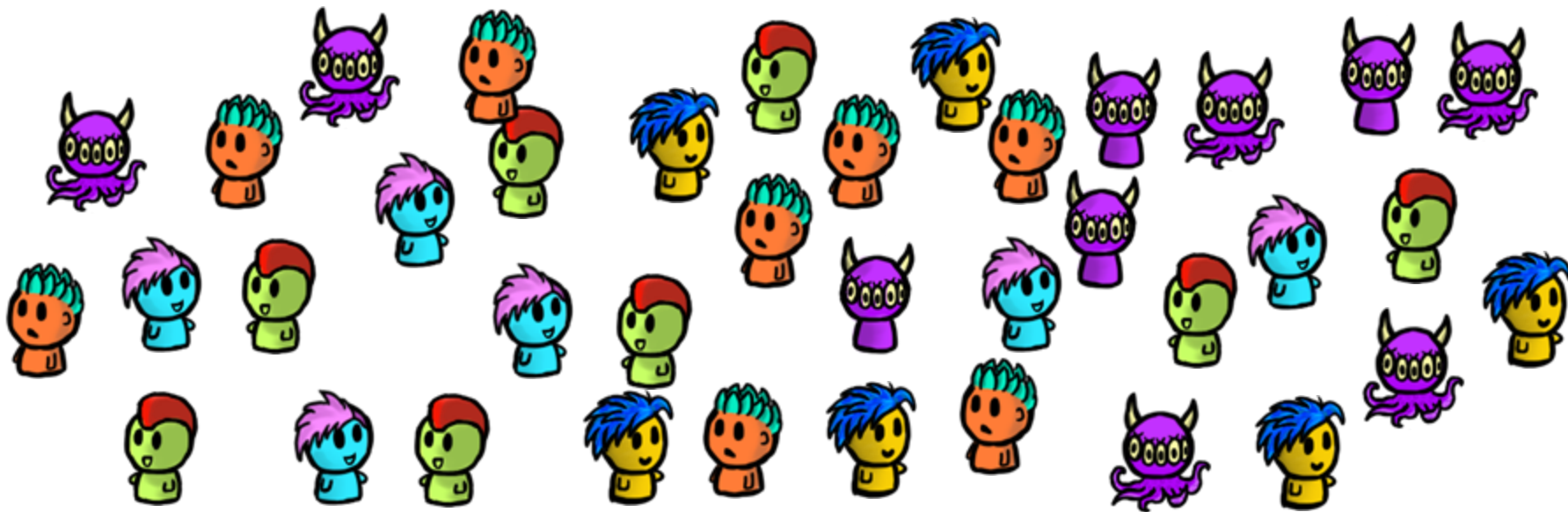
What's the probability two of us have the same birthday?

There's 23 of us, so larger than **50%!!**



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 60 of us, it's more than 99%!!!

Collisions and the Birthday Paradox

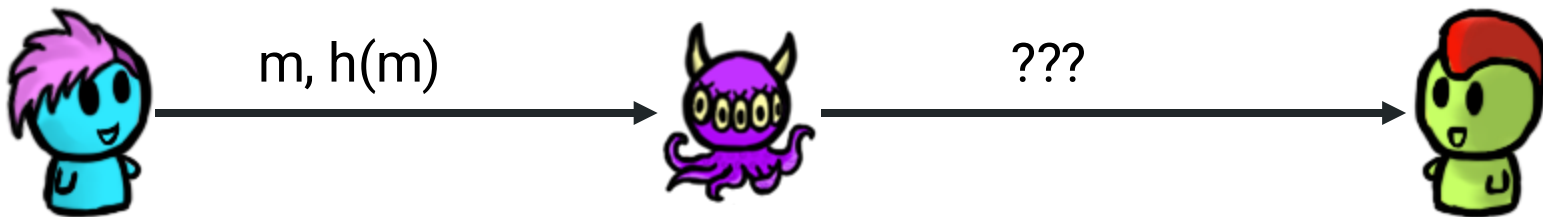
Collisions are easy due to

This is NOT the end of
our problems...

9%!!!

How about a bad example?

Assume we **don't care** about confidentiality now, just integrity.



Q: What can Mallory do to send the message she wants (i.e., **change m**)?

How about a bad example?

Assume we **don't care** about confidentiality now, just integrity.



Q: What can Mallory do to send the message she wants (i.e., **change m**)?

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



How about a bad example?

Assume we also **care** about confidentiality now.



Q: What can Mallory do to send the message she wants (i.e., **change** $E(m)$)?

How about a bad example?

Assume we also **care** about confidentiality now.



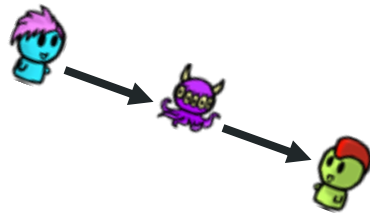
Q: What can Mallory do to send the message she wants (i.e., **change $E(m)$**)?

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

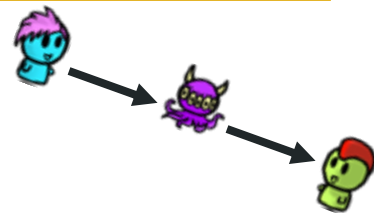


I could publish
the hash of my
public key on a
business card



Limitations for Cryptographic Hash Functions

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



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



Good idea! Although the key would be **too big** to place on the card, I could use the hash to... verify it!

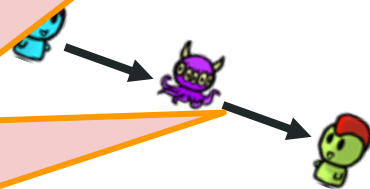
Limitations for Cryptographic Hash Functions

- Integrity guarantees only when using a secure way of sending/storing the data

What if...we don't have an external/physical channel?
i.e., using the Internet to communicate

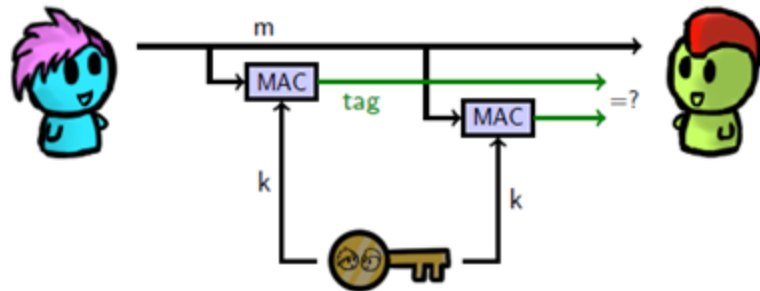
I could publish the hash of my public key on my business card

the key would be on a card, I could use the hash to... verify it!



Authentication and Hash Functions

- We can use “keyed hash functions”
- Requires a **secrete** key to generate, or even check, the computed hash value (sometimes called a **tag**)



Called: Message authentication codes (MACs)

Authentication and Hash Functions

- We can use “keyed hash functions”

Requires a **secrete** key to generate, or even check, the computed hash value

(sometimes called a **tag**)

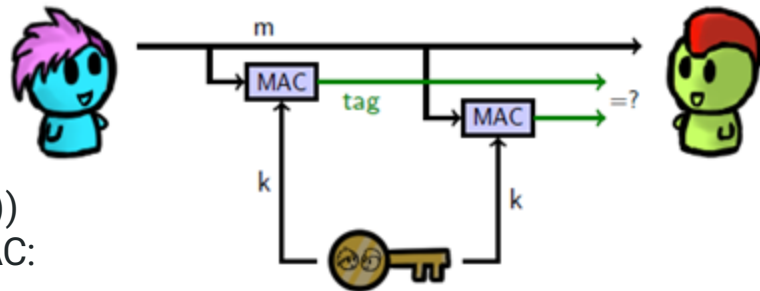
→ Need to exchange the **secrete** key (D.H/PKC)

→ $\text{HMAC}(K, M) = \text{Hash}((K \oplus \text{opad}) \parallel \text{Hash}((K \oplus \text{ipad}) \parallel M))$

opad and **ipad** are fixed constant values used in HMAC:

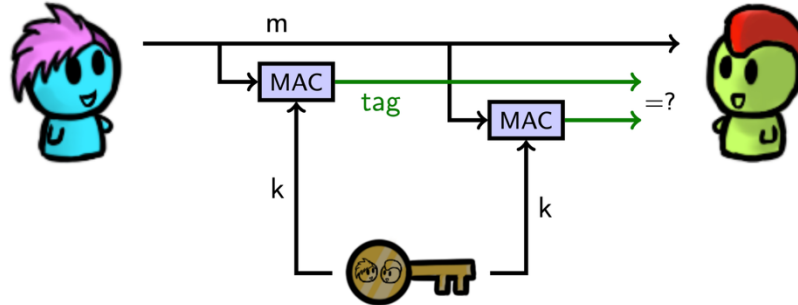
ipad = 0x36 repeated 64 times (inner pad)

opad = 0x5C repeated 64 times (outer pad)



Called: Message authentication codes (MACs)

Message Authentication Codes (MACs)



Do the MAC/tag values match?

YES

NO

No one
messed with
the data

The data has
been altered
somehow

I don't have the key
to generate or
check the values...

Common examples:

- SHA-1-HMAC, SHA-256-HMAC, CBC-MAC

Combine Ciphers and MACs



Confidentiality



Integrity

Combine Ciphers and MACs



Confidentiality



Integrity

In practical we often need both
confidentiality and message integrity

But how to combine them? Three possibilities

There are multiple strategies to combine a cipher and a MAC when processing a message

MAC-then-Encrypt,

Encrypt-and-MAC,

Encrypt-then-MAC

But how to combine them? Three possibilities

There are multiple strategies to combine a cipher and a MAC when processing a message

MAC-then-Encrypt,

Encrypt-and-MAC,

Encrypt-then-MAC

Ideally crypto libraries already provides an **authenticated encryption mode** that securely combines the two operations, so we don't have to worry about getting it right

- E.g., GCM, CCM (used in WPA2, see later), or OCB mode

Let's try it!

- Alice and Bob have a secret key **K** for a cryptosystem $(E_k(\cdot), D_k(\cdot))$
- Also, a secret key **K'** for their $MAC_{K'}(\cdot)$



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

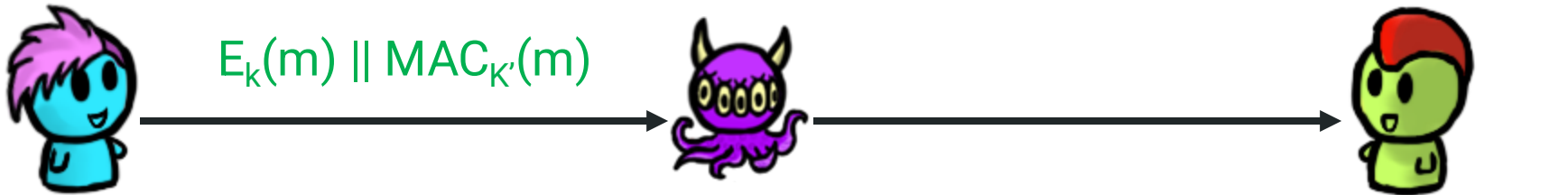
MAC-then-Encrypt

- Compute the MAC on the message, then encrypt the message and MAC together, and send that ciphertext.



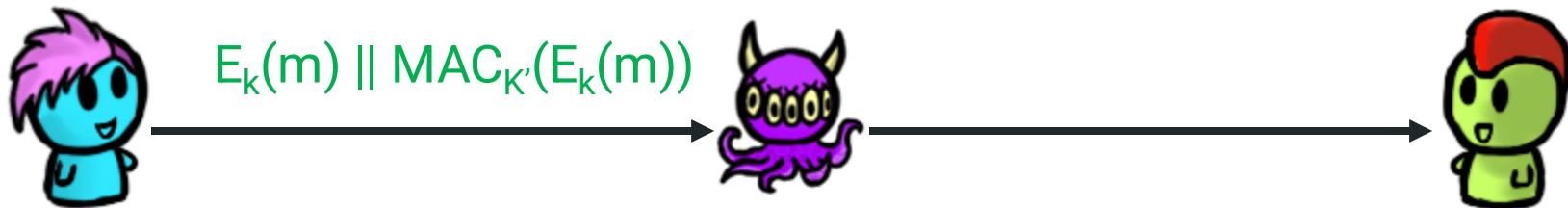
Encrypt-and-MAC

- Compute the MAC on the message, the encryption of the message, and send both.



Encrypt-then-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?

$E_k(m \parallel \text{MAC}_{K'}(m))$ **vs.** $E_k(m) \parallel \text{MAC}_{K'}(m)$ **vs.** $E_k(m) \parallel \text{MAC}_{K'}(E_k(m))$

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**

The Doom of MAC-then-Encrypt $E_k(m \parallel \text{MAC}_{K'}(m))$



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.

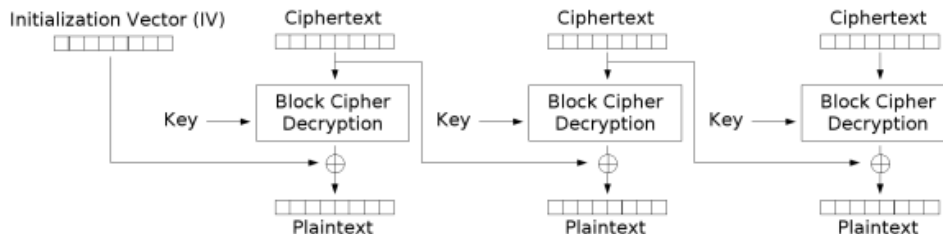


The Doom of MAC-then-Encrypt

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

- **Padding oracle attack:**

- So if a block needs to be padded out by 5 bytes, for instance, one would append 5 bytes of the value 0x05.
- 1st decrypt the message, look at the value of the last byte (call it N), and then insure that the preceding N-1 bytes also had the value of N.
- If we encounter an incorrect value → padding error, and should abort. Since the MAC is part of the encrypted payload, *all* of this needs to happen before the MAC can be verified.



Cipher Block Chaining (CBC) mode decryption



The Doom of MAC-then-Encrypt

CBC encrypts data in fixed-size blocks (usually 16 bytes). We might need **padding**.

- If you need 1 byte of padding: add 01
- If you need 2 bytes of padding: add 02 02
- If you need 5 bytes of padding: add 05 05 05 05 05

Example: Padding in Action

- Original message: "Hello World!"
- Message length: 12 bytes
- Block size: 16 bytes
- Padding needed: 4 bytes
- Padded message: "Hello World!" + 04 04 04 04

Normal CBC Decrypt Process

- Decrypt the ciphertext
- Remove padding (check if valid=**Critical Flaw**)
 - "Padding Error" = The decrypted byte doesn't match expected padding
 - "Valid Padding" = We've correctly guessed what that byte should be
- Verify MAC/signature
- Process the message



The Doom of MAC-then-Encrypt

Example: Padding in Action

- Padded message: "Hello World!" + HMAC (32 bytes) + [0x04, 0x04, 0x04, 0x04] (padding)
- Encrypt with CBC: IV + C_1 + C_2 + C_3

Setup: The attacker targets the last block C_3 using the previous block C_2

Attack Strategy: Modify C_2 to control how C_3 decrypts

- $\text{Plaintext}_3 = \text{Decrypt}(C_3) \oplus C_2$
- By changing C_2 to C_2' , we get: $\text{Plaintext}_3' = \text{Decrypt}(C_3) \oplus C_2'$

Byte Recovery Process: Start with byte 15 of P3 (last byte, contains padding)

- Try all 256 values for $C_2'[15]$ until oracle returns "valid padding"
- When found, calculate: $\text{Plaintext}_3 = C_2[15] \oplus C_2'[15] \oplus 0x01$
- Move to byte 14, create padding 0x02 0x02
- Repeat for all bytes
- We recover the last 12 bytes of HMAC and the padding [0x04, 0x04, 0x04, 0x04]
- Attack the remaining Blocks
 - Use C_2 and C_1 to attack Block 2 (more MAC bytes)
 - Use C_1 and IV (if available) to attack Block 1 ("Hello World!" + start of MAC)



The Doom Principle $E_k(m) \parallel \text{MAC}_{K'}(m)$

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

- **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 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 $E_k(m) \parallel \mathbf{MAC}_k(\mathbf{m})$ will not leak information about the secret message \mathbf{m} .

Which order is correct?

We want the receiver to verify the MAC first!

The recommended strategy is Encrypt-then-MAC:
 $E_k(m) \parallel \text{MAC}_{K'}(E_k(m))$

- **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)

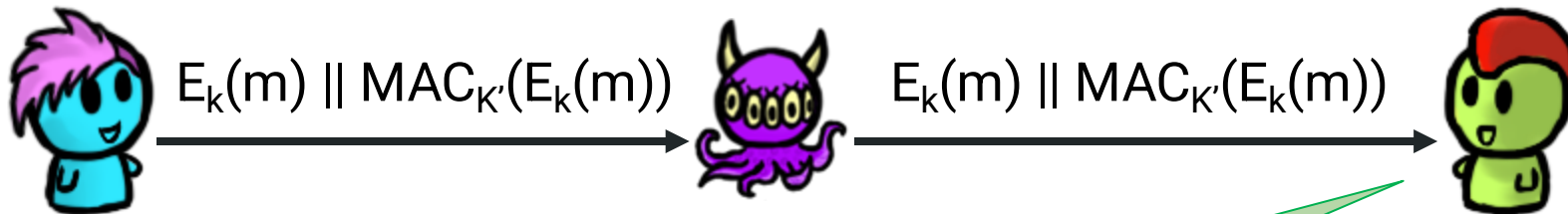
Sweet!





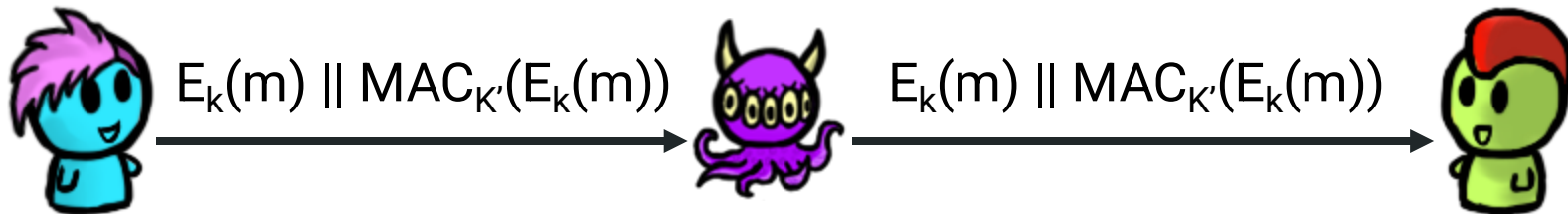
More properties that matter?

Repudiation



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

Repudiation



Confidentiality

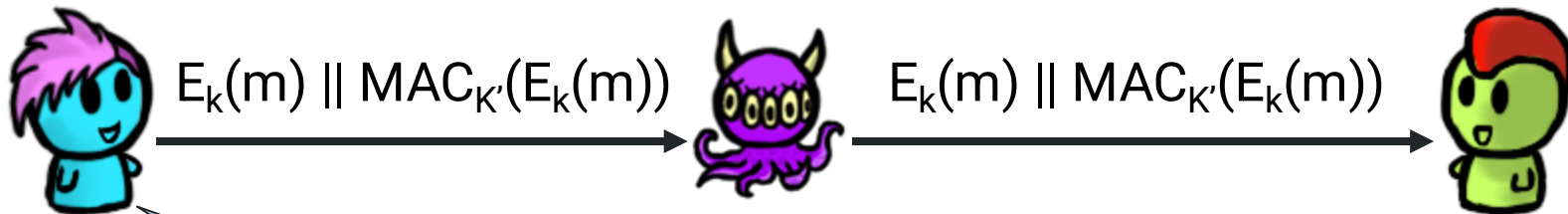


Integrity



Authentication

Repudiation



Almost, but not quite a “signature”



Confidentiality

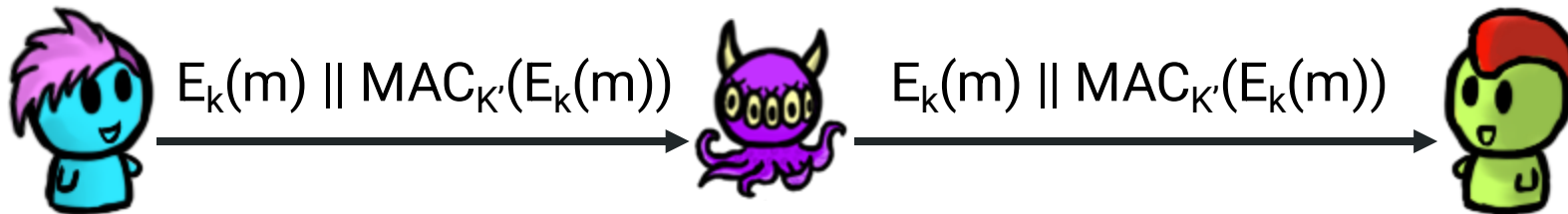


Integrity



Authentication

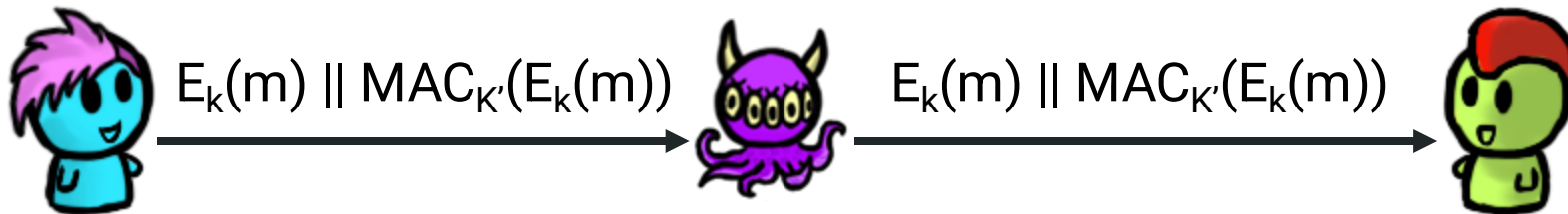
Repudiation



So...you're saying Bob can't prove to Carol that Alice sent m?



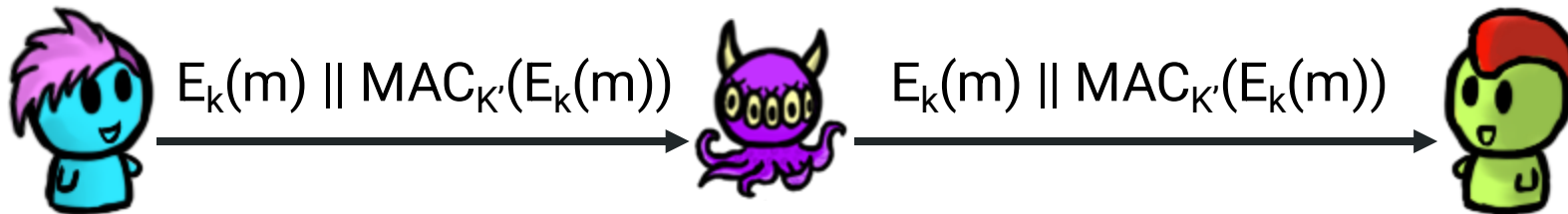
Repudiation



So...you're saying Bob can't prove to Carol that Alice sent m?

Q: Why can't Bob prove it?

Repudiation

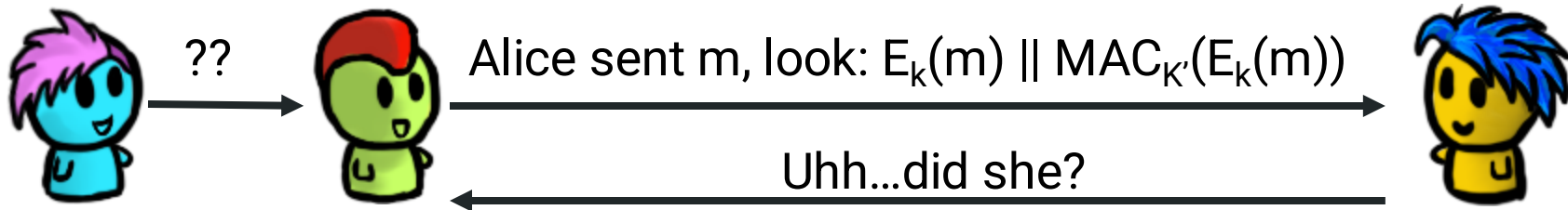


So...you're saying Bob can't prove to Carol that Alice sent m?

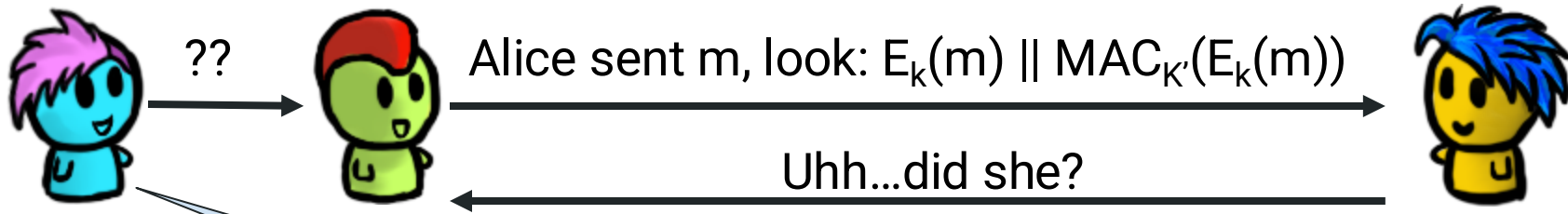
Q: Why can't Bob prove it?

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

Implications?



Implications?

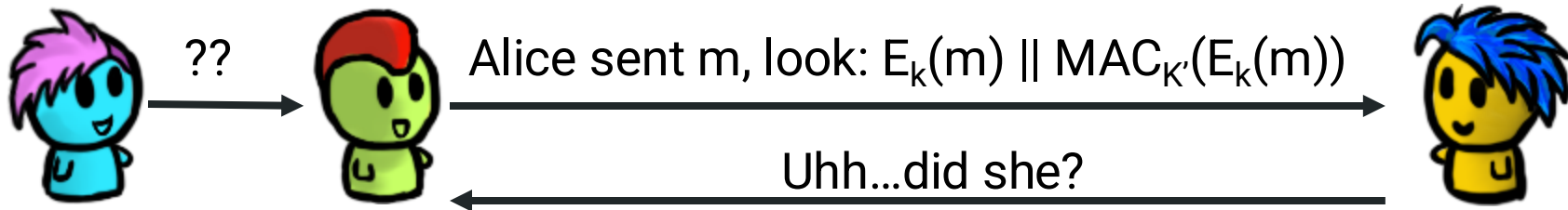


Nope! Bob made everything up!
Both the message and the MAC



Bob be like

Implications?

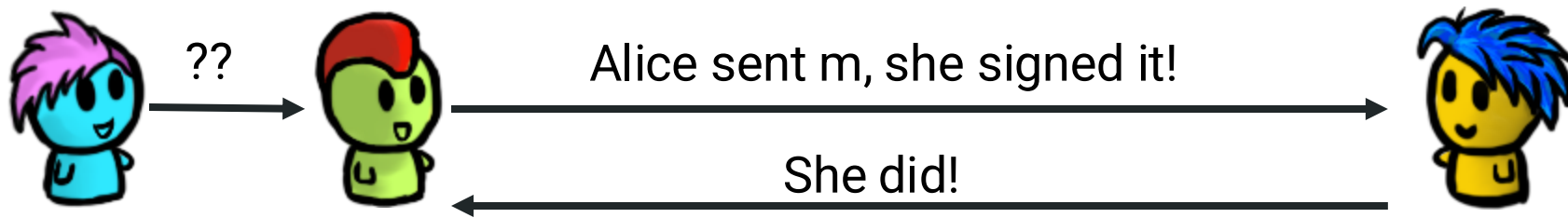


This is called **repudiation**, and we sometimes want to avoid it

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

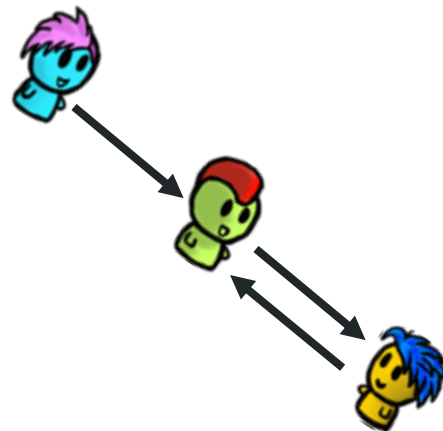
Digital Signatures - For When Repudiation is Bad

For **non-repudiation**, what we want is a true **digital signature**, with the following properties:




Properties of digital signatures

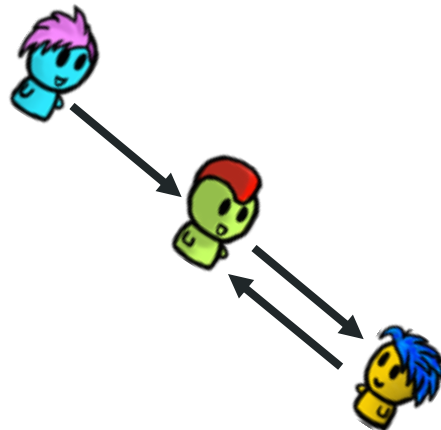
If Bob receives a message with Alice's digital signature on it, then:



Properties of digital signatures

If Bob receives a message with Alice's digital signature on it, then:

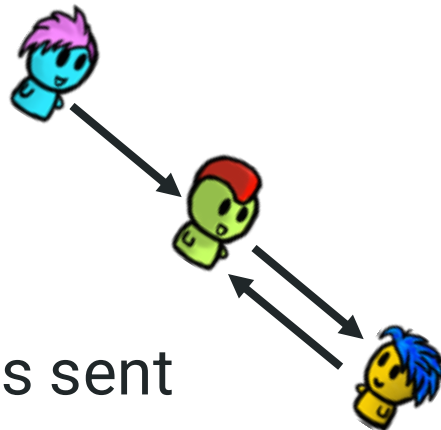
- Alice sent it, and not , (like a MAC)



Properties of digital signatures


If Bob receives a message with Alice's digital signature on it, then:

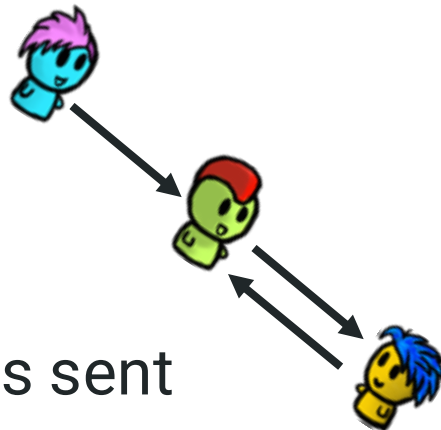
- Alice sent it, and not 🐉, (like a MAC)
- The message has not been altered since it was sent (like a MAC)



Properties of digital signatures

If Bob receives a message with Alice's digital signature on it, then:

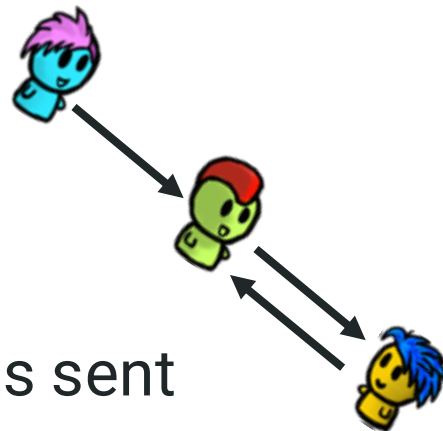
- Alice sent it, and not , (like a MAC)
- The message has not been altered since it was sent (like a MAC)
- Bob can prove these properties to a third party (NOT like a MAC)



Properties of digital signatures

If Bob receives a message with Alice's digital signature on it, then:

- Alice sent it, and not 🧛, (like a MAC)
- The message has not been altered since it was sent (like a MAC)
- Bob can prove these properties 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 Alice public verification key



3. Alice signs m with her private **signature key** S_k

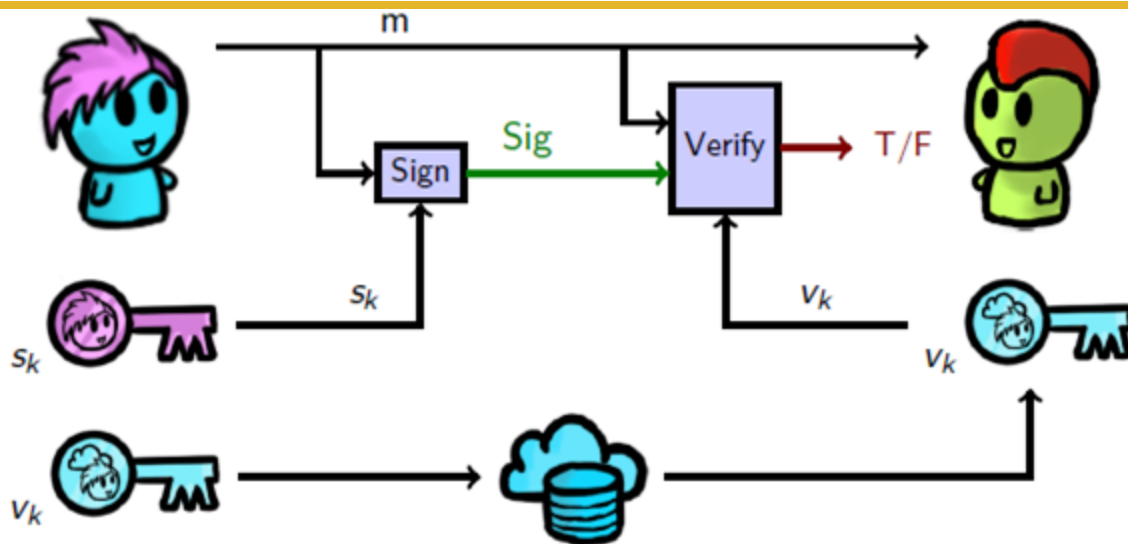


4. Bob verifies m with Alice's public **verification key** V_k



5. If it verifies correctly, the signature is valid

Digital Signatures at a Glance



Extending the concept of H-MAC to scenarios where we need non-repudiation/don't have shared secrets.

Digital Signatures at a Glance

Example of Signature Verification

Message: "Hello World!"

Hash: $h = \text{SHA-256}(\text{"Hello World!"}) = 665$ (simplified)

Private key: $d = 2753$ (this is the secret)

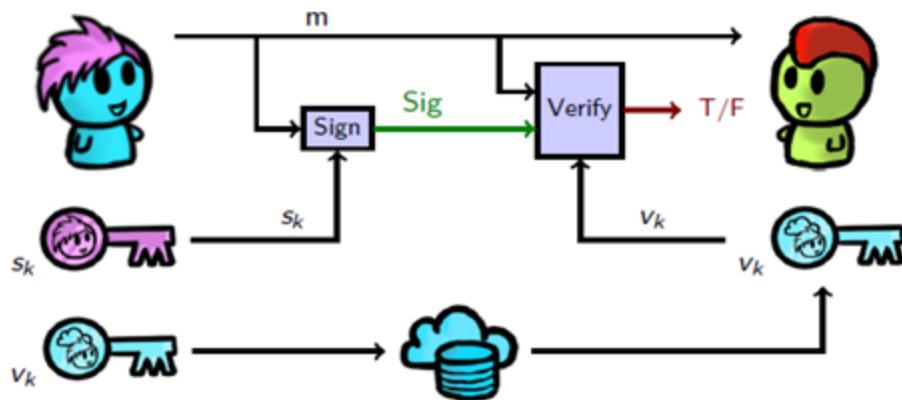
Modulus: $n = 3233$

Signature: $\sigma = h^d \bmod n = 665^{2753} \bmod 3233 = 1206$

Public key components: $e = 17, n = 3233$

Verification: $\sigma^e \bmod n = 1206^{17} \bmod 3233 = 665$

Check: $665 == \text{original hash}$ ✓

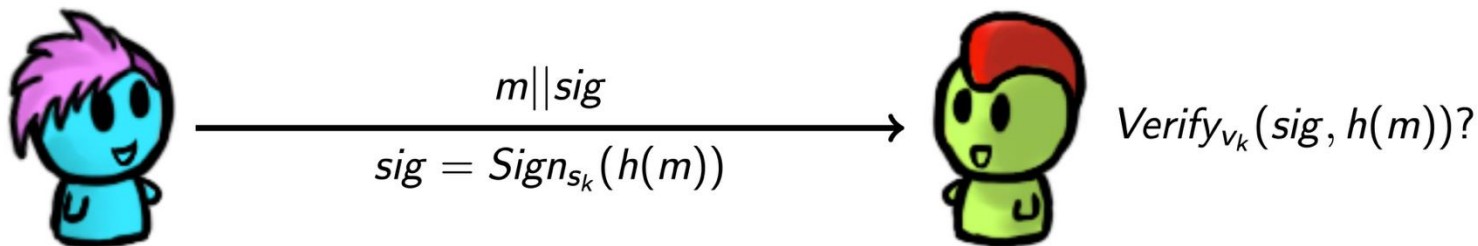


Faster Signatures

- Signing large messages is slow
→ “hybridize” the signatures to make them faster
- A hash is much smaller than the message... faster to sign

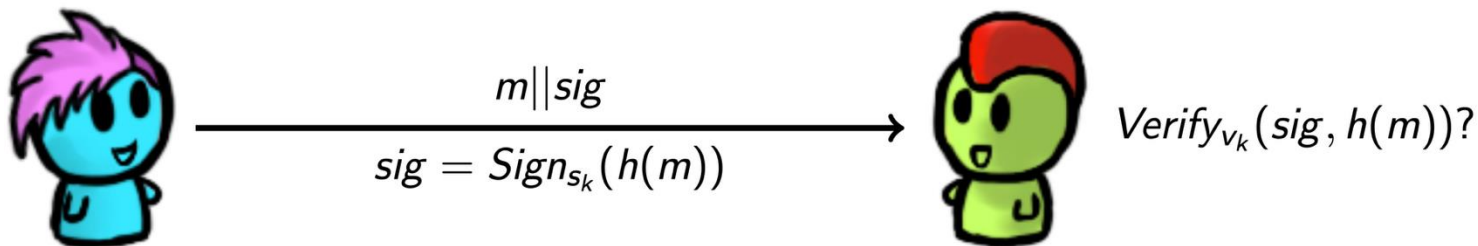
Faster Signatures - aka More Hybrids

- Signing large messages is slow
→ “hybridize” the signatures to make them faster
- A hash is much smaller than the message... faster to sign



Faster Signatures - aka More Hybrids

- Signing large messages is slow
→ “hybridize” the signatures to make them faster
- A hash is much smaller than the message... faster to sign



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

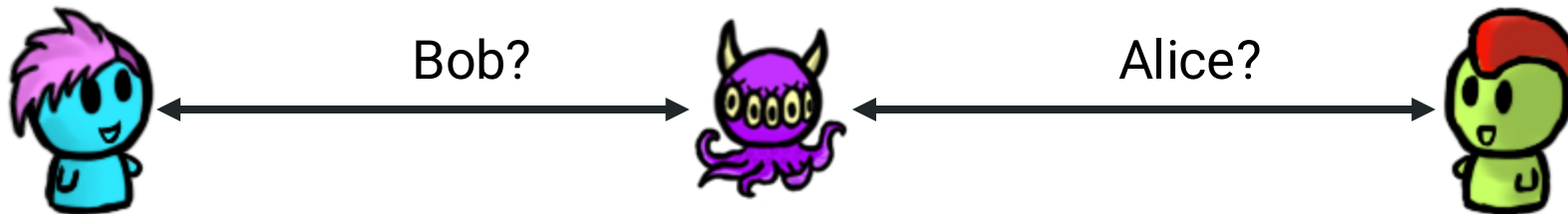
Combining PKE and digital signatures

- Alice has two different key pairs (Security best practices require separation):
 - An (encryption, decryption) key pair e_k^A, d_k^A
 - An (signature, verification) key pair s_k^A, v_k^A
- So does Bob : e_k^B, d_k^B and s_k^B, v_k^B
- Alice uses e_k^B to encrypt a message destined for Bob:
 - $C = E_{e_k^B}(M)$
- She uses s_k^A to sign the ciphertext:
 - $T = \text{Sign}_{s_k^A}(C)$
- Bob uses v_k^A to check the signature:
 - $\text{Verify}_{v_k^A}(C, T)$, if verified, C is authentic
- He uses d_k^B to check the ciphertext:
 - $M = D_{d_k^B}(C)$

Relationship between key pairs

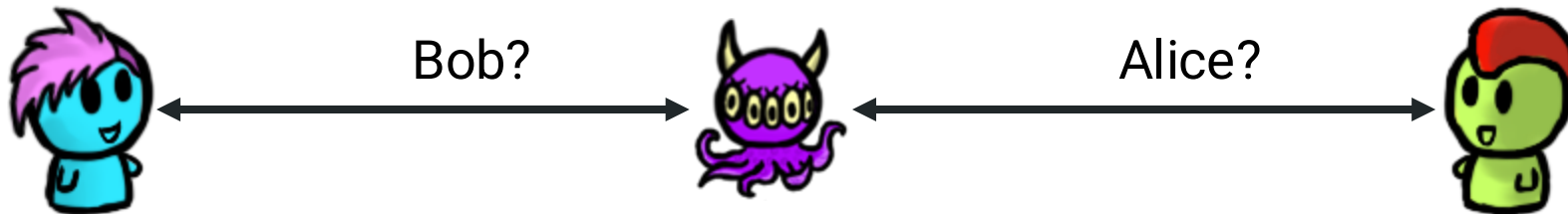
- Alice (signature, verification) key pair is long-lived, whereas her (encryption, decryption) key pair is short-lived
 - Provides forward secrecy
- When creating a new (encryption, decryption) key pair, Alice uses her signing key to sign her new encryption key and Bob uses Alice's verification key to verify the signature on this new key

The Key Management Problem



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

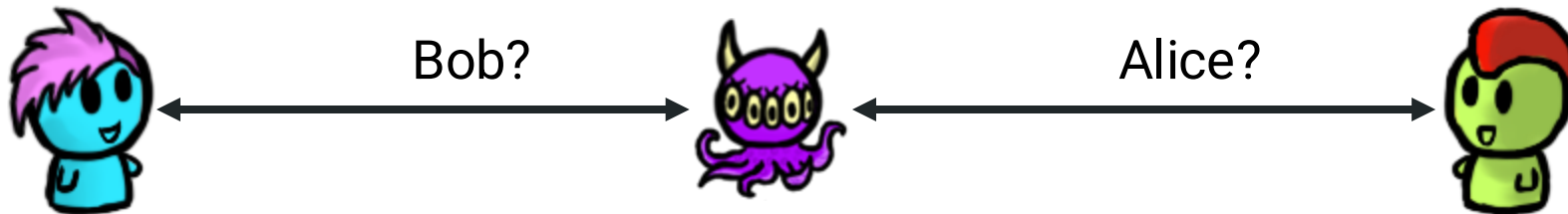
The Key Management Problem



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

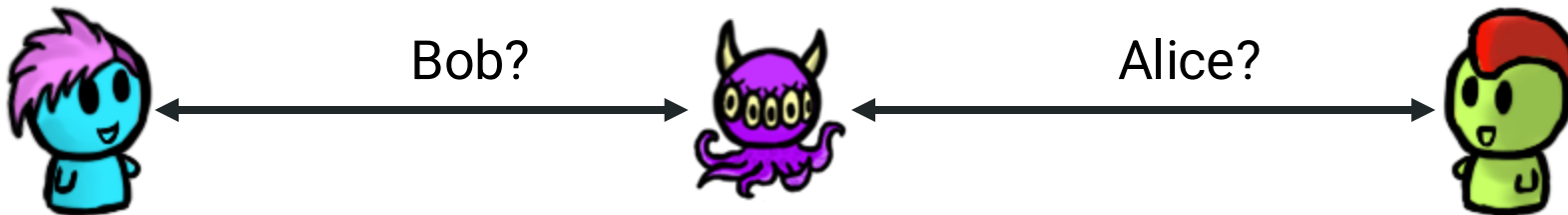


Q: How can Alice and Bob be sure they're talking to each other?

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...

Symmetric

Ciphers

**Hash
Functions**

**Message
Auth. codes**

PRFs

Stream

Block

Asymmetric

PKE

**Digital
Signatures**

**Key
Exchange**

RSA

IND-CCA security types

Discrete Log...