hashes, MACs, and diffie-hellman

slides
bit.ly/cs161-disc

feedback
bit.ly/extended-feedback

hack of the day

hack of the day

- secure messaging app Threema was found to “feature neither ‘forward security’ or post-compromise security”

hack of the day

- secure messaging app Threema was found to “feature neither ‘forward security’ or post-compromise security”
    - “…compromising a single client ephemeral key allowed an attacker to impersonate that client indefinitely”

hack of the day

- secure messaging app Threema was found to “feature neither ‘forward security’ or post-compromise security”
  - “…compromising a single client ephemeral key allowed an attacker to impersonate that client indefinitely”
  - can decrypt previous communications as well

hack of the day

- secure messaging app Threema was found to “feature neither ‘forward security’ or post-compromise security”
    - “...compromising a single client ephemeral key allowed an attacker to impersonate that client indefinitely”
    - can decrypt previous communications as well
    - problem with developing your own crypto

general questions, concerns, etc.

skip to diffie-hellman?

reminder: cryptography

- why?
    - secure communication
- goals:
    - confidentiality: adversary cannot <u>read</u> messages
    - integrity: adversary cannot <u>change</u> messages
    - authenticity: message is from the claimed author

picking up from last time

picking up from last time

- we saw how to ensure **confidentiality**

picking up from last time

- we saw how to ensure **confidentiality**
  - block ciphers + modes of operation
  - IND-CPA, CBC, CTR

picking up from last time

- we saw how to ensure **confidentiality**
  - block ciphers + modes of operation
  - IND-CPA, CBC, CTR
- what if mallory (someone who can manipulate messages sent over a channel) tampers with the message?

picking up from last time

- we saw how to ensure **confidentiality**
  - block ciphers + modes of operation
  - IND-CPA, CBC, CTR
- what if mallory (someone who can manipulate messages sent over a channel) tampers with the message?
- looking to also enforce **integrity** and **authenticity**

cryptographic hashes

cryptographic hashes

- a fast, deterministic way to produce a seemingly random output of bits given an input

cryptographic hashes

- a fast, deterministic way to produce a seemingly random output of bits given an input
- H(M): M is an arbitrary length message
  - output: <u>fixed length</u> n-bit hash
  - $\{0, 1\}^* \rightarrow \{0, 1\}^n$

cryptographic hashes

- a fast, deterministic way to produce a seemingly random output of bits given an input
- H(M): M is an arbitrary length message
  - output: <u>fixed length</u> n-bit hash
  - $\{0, 1\}^* \rightarrow \{0, 1\}^n$
- properties

cryptographic hashes

- a fast, deterministic way to produce a seemingly random output of bits given an input
- H(M): M is an arbitrary length message
    - output: <u>fixed length</u> n-bit hash
    - $\{0, 1\}^* \rightarrow \{0, 1\}^n$
- properties
    - correctness, efficiency, security—remember these?

hashes: one-way-ness

hashes: one-way-ness

- a.k.a preimage resistance
- given output y, infeasible to find <u>any</u> x s.t. $H(x) = y$

hashes: one-way-ness

- a.k.a preimage resistance
- given output y, infeasible to find <u>any</u> x s.t. $H(x) = y$
- is $H(x) = x^3$ one way?

hashes: one-way-ness

- a.k.a preimage resistance
- given output y, infeasible to find <u>any</u> x s.t. $H(x) = y$
- is H(x) = x^3 one way?
  - no!

hashes: one-way-ness

- a.k.a preimage resistance
- given output y, infeasible to find <u>any</u> x s.t. H(x) = y
- is H(x) = x^3 one way?
  - no!
  - e.g., given output y, one can take cube root of y to find an x

hashes: collision resistance

hashes: collision resistance

- collision: two different inputs have the same output

hashes: collision resistance

- collision: two different inputs have the same output
- collision resistance: infeasible to find a pair of different inputs x, x' s.t. $H(x) = H(x')$

hashes: collision resistance

- collision: two different inputs have the same output
- collision resistance: infeasible to find a pair of different inputs x, x’ s.t. H(x) = H(x’)
- is H(x) = x^2 collision resistant?

hashes: collision resistance

- collision: two different inputs have the same output
- collision resistance: infeasible to find a pair of different inputs x, x’ s.t. H(x) = H(x’)
- is H(x) = x^2 collision resistant?
  - no!

hashes: collision resistance

- collision: two different inputs have the same output
- collision resistance: infeasible to find a pair of different inputs x, x’ s.t. H(x) = H(x’)
- is H(x) = x^2 collision resistant?
  - no!
  - x = 1, x’ = -1 both hash to 1

hashes: randomness

hashes: randomness

- say H(161) = 619470

hashes: randomness

- say H(161) = 619470
  - looks random to me, right?

hashes: randomness

- say H(161) = 619470
  - looks random to me, right?
- what if H(162) = 619471

hashes: randomness

- say H(161) = 619470
  - looks random to me, right?
- what if H(162) = 619471
  - the hash looks pretty similar to the one before

hashes: randomness

- say H(161) = 619470
  - looks random to me, right?
- what if H(162) = 619471
  - the hash looks pretty similar to the one before
- a truly pseudorandom hash function means flipping one bit in the input generates an observably random output

hashes: randomness

- say H(161) = 619470
  - looks random to me, right?
- what if H(162) = 619471
  - the hash looks pretty similar to the one before
- a truly pseudorandom hash function means flipping one bit in the input generates an observably random output
- is the example hash secure?

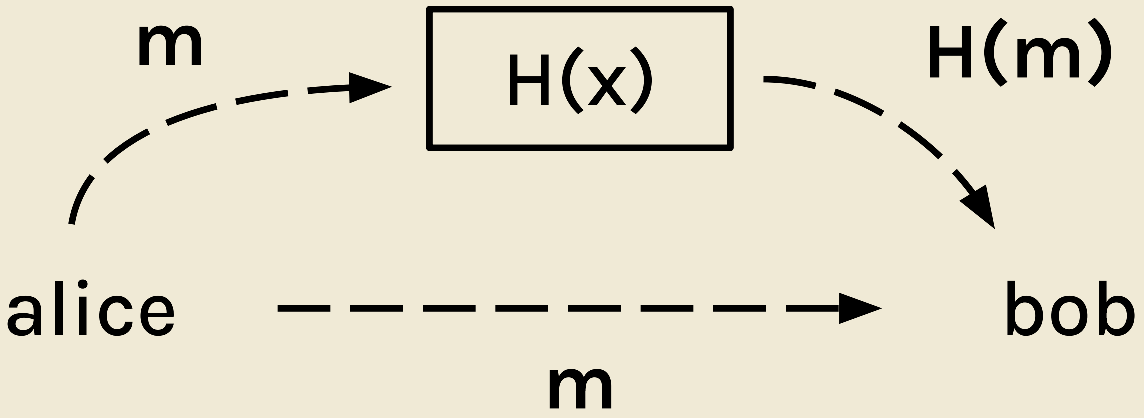
hashes in real life

- MD5 (128 bits): broken
- SHA-1 (160 bits): broken
- SHA-2 (256, 384, 512 bits): some variants vulnerable to length extension attack
- SHA-3/Keccak (256, 384, 512 bits): current standard

hashes and integrity

- can hashes provide integrity?
  - if the sent hash remains unmodified

hashes and integrity



if the message is tampered with (and H(m) isn't), Bob can compute H(m) himself and make sure it matches the sent hash

hashes and integrity

hashes and integrity

- back to the question: can hashes provide integrity?

hashes and integrity

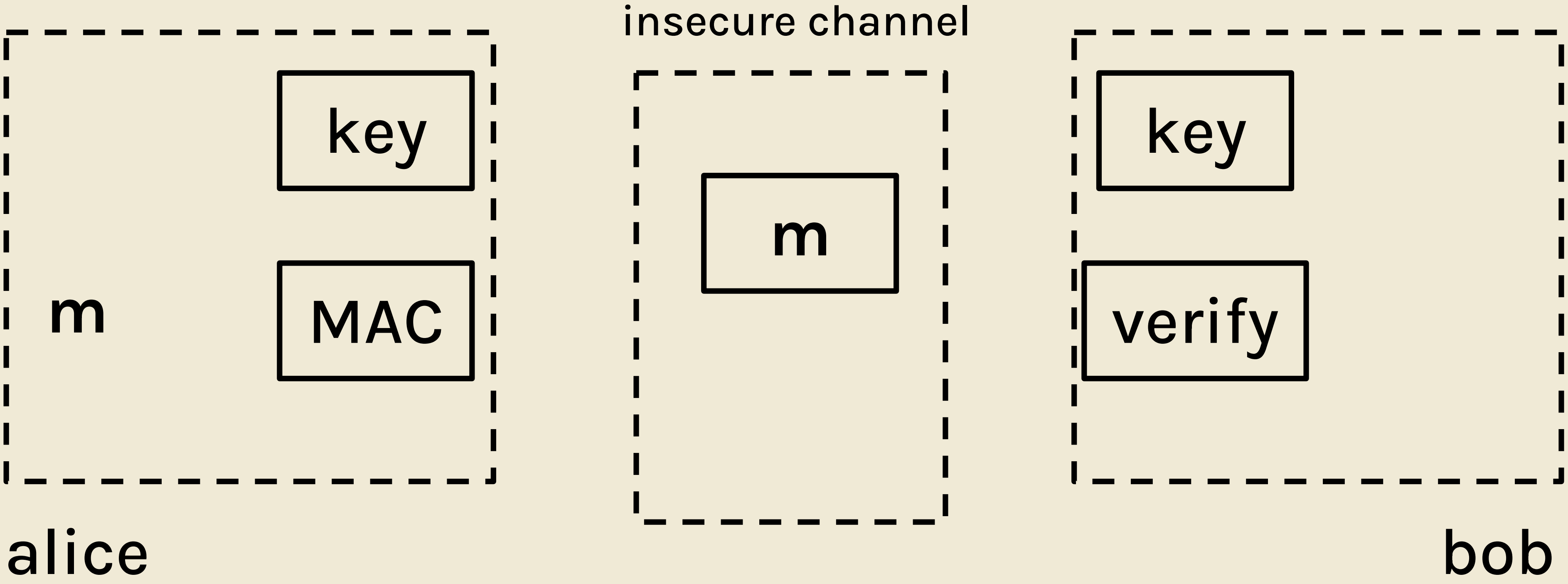
- back to the question: can hashes provide integrity?
  - not alone, because hashes are *unkeyed*—anyone can compute the hash of any plaintext

hashes and integrity

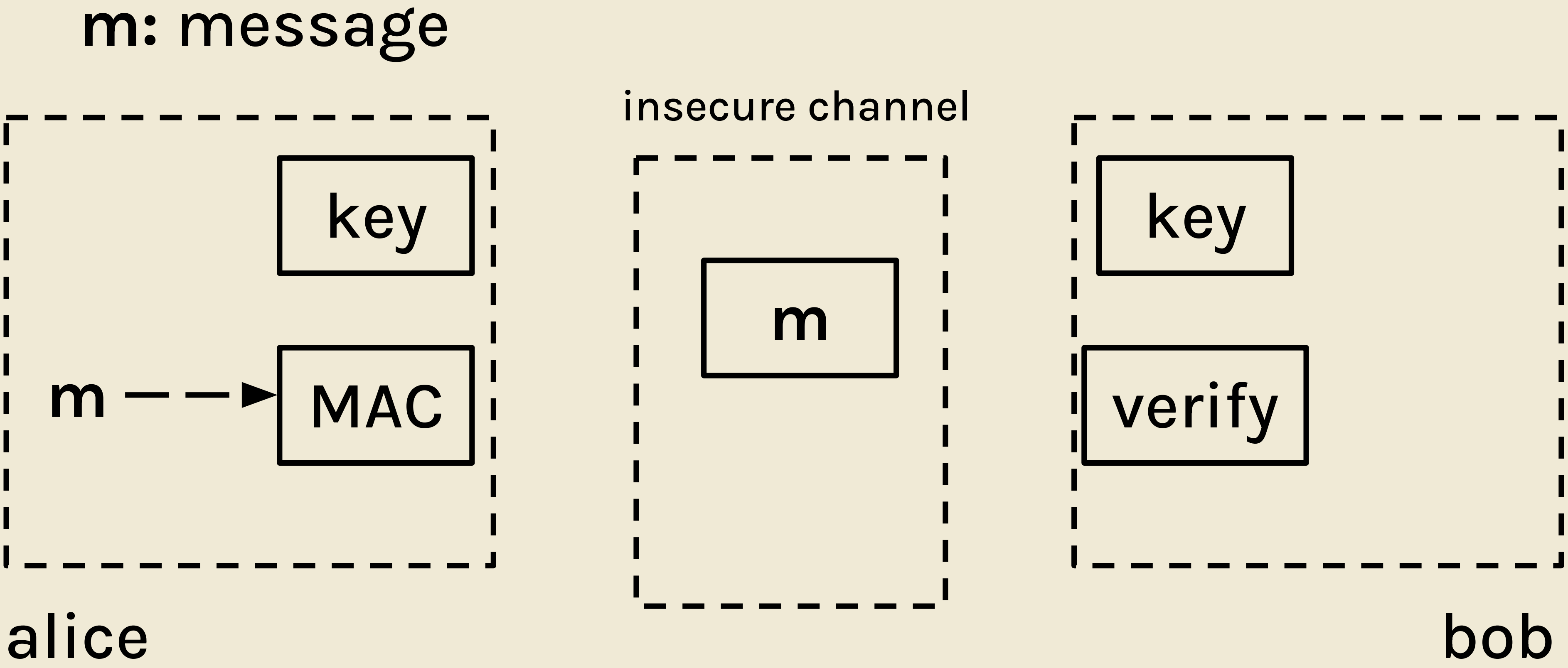
- back to the question: can hashes provide integrity?
  - not alone, because hashes are *unkeyed*—anyone can compute the hash of any plaintext
  - introducing MACs

message authentication codes

m: message

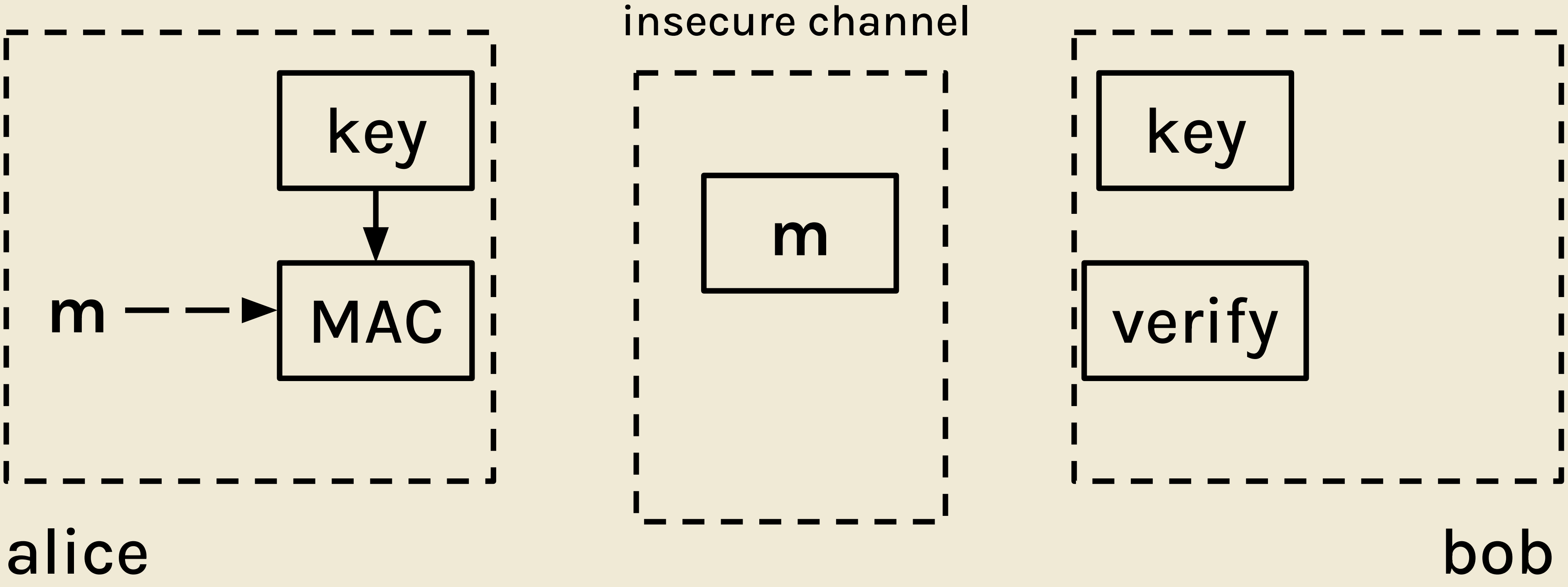


message authentication codes



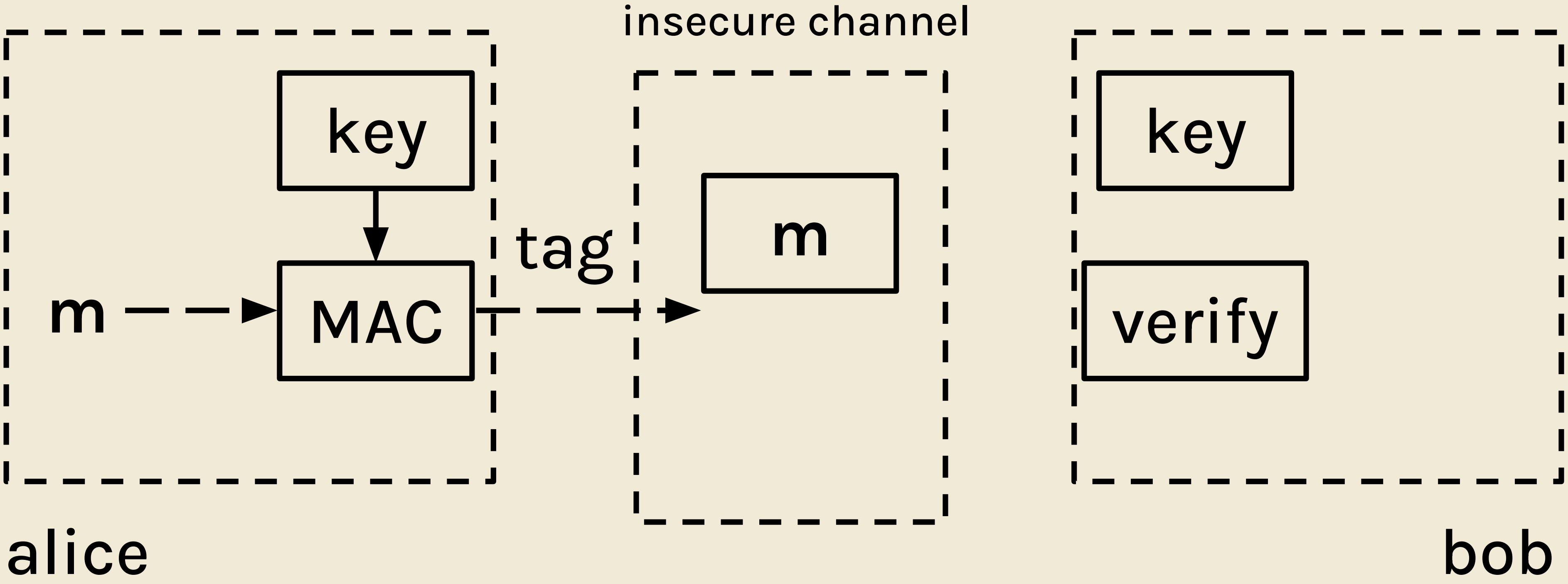
message authentication codes

m: message



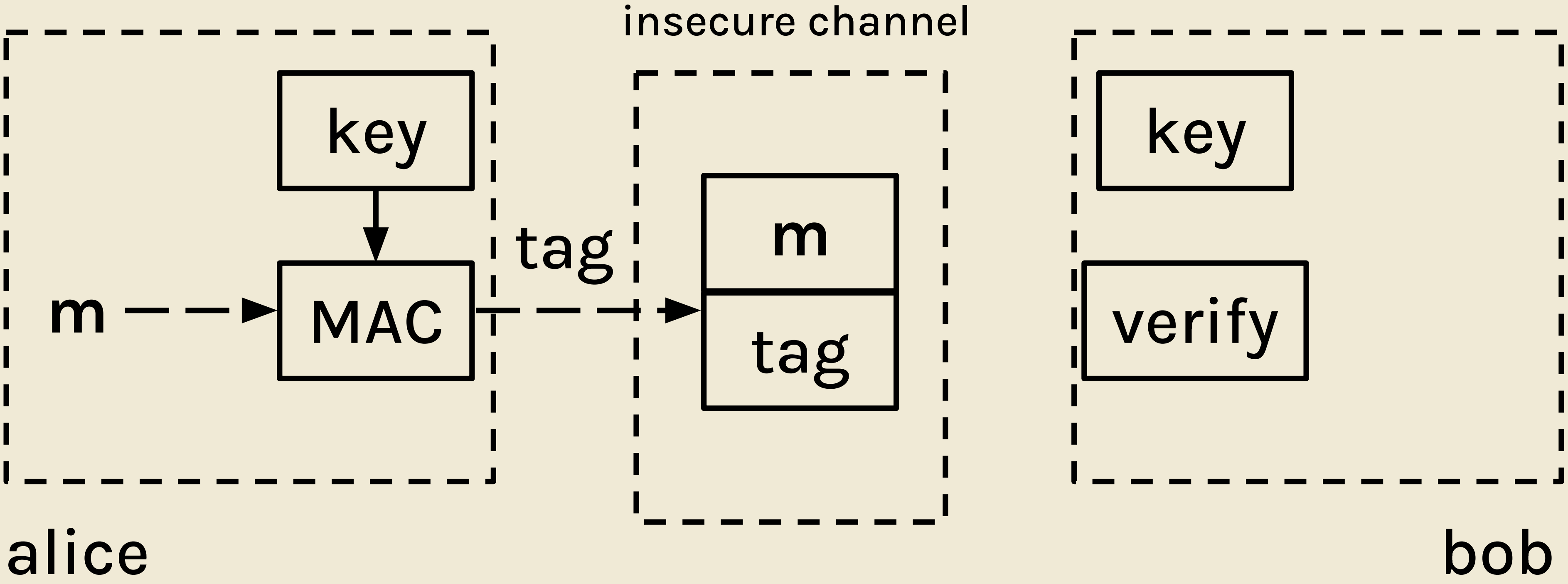
message authentication codes

m: message



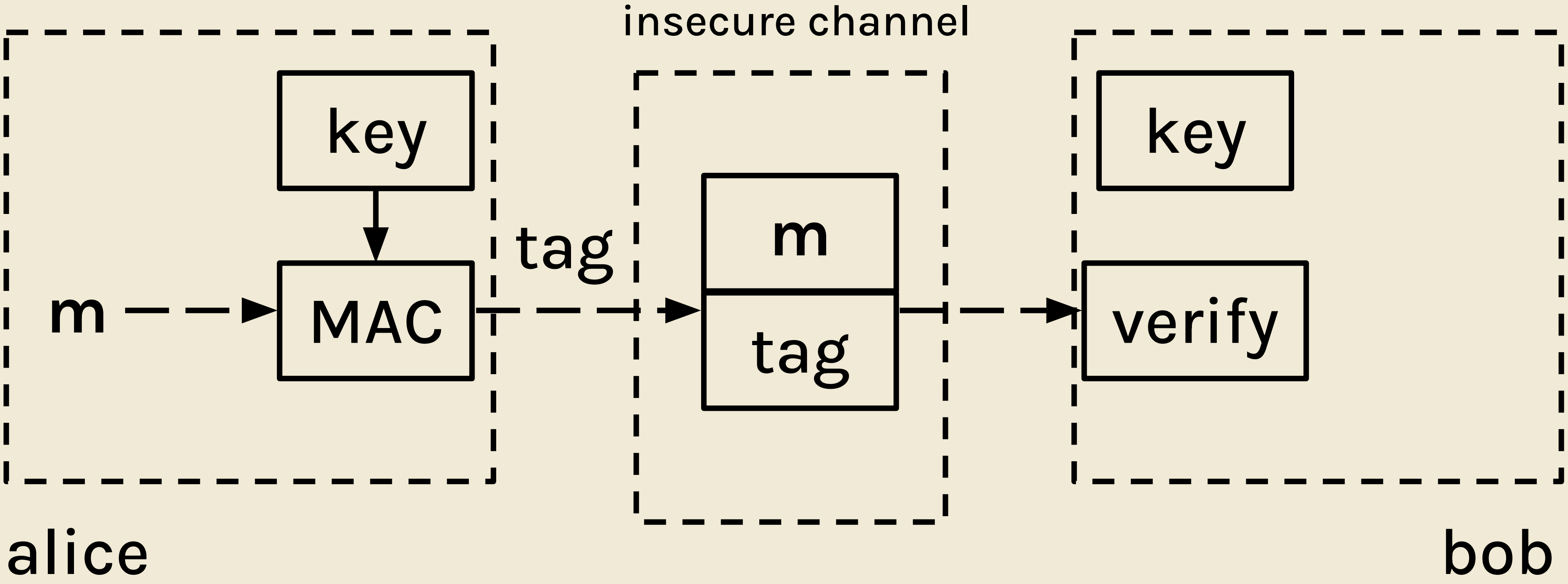
message authentication codes

m: message



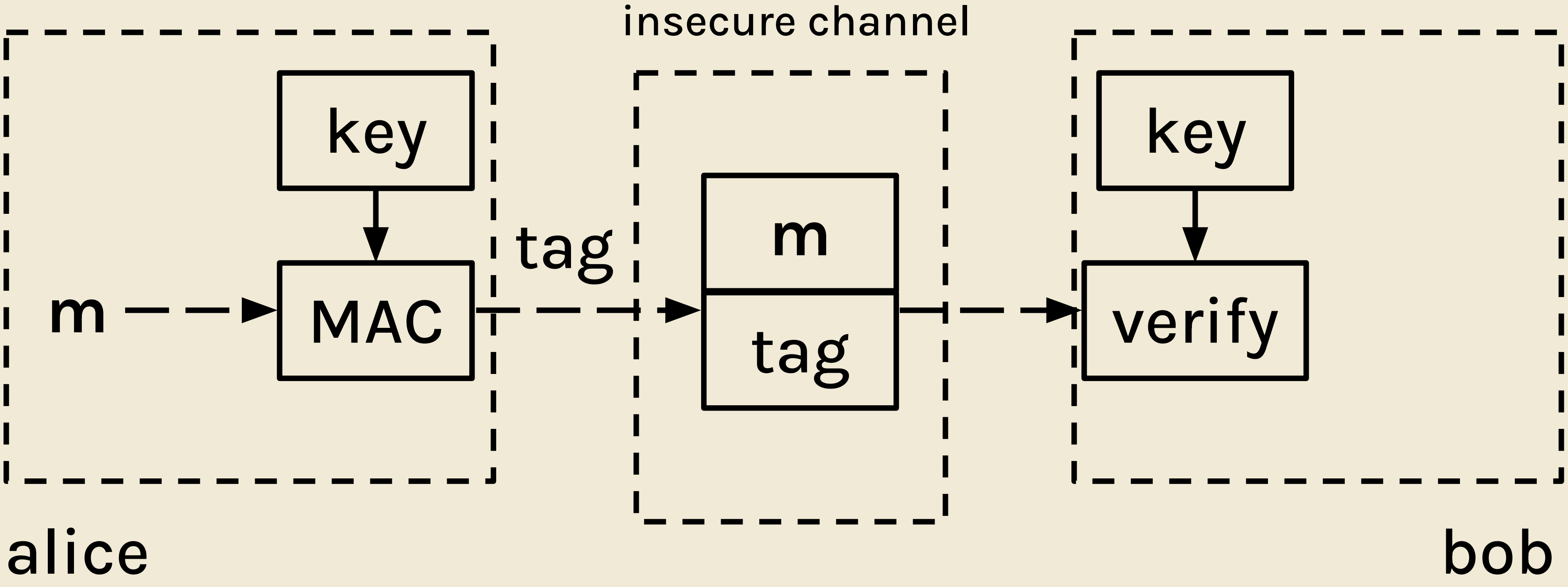
message authentication codes

m: message



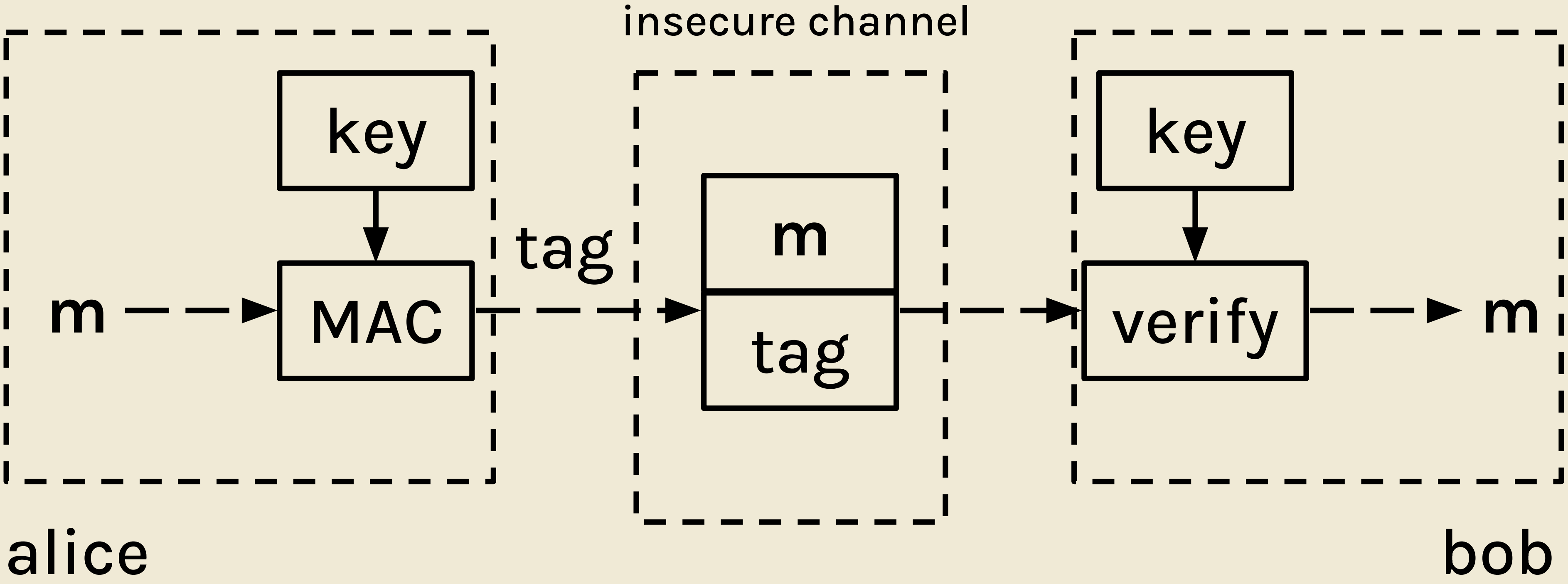
message authentication codes

m: message



message authentication codes

m: message



message authentication codes (MAC)

- KeyGen() $\rightarrow K$: generate a key K
- MAC(K, M) $\rightarrow T$: generate tag T for message M using key K
  - inputs: secret key and arbitrary-length message
  - output: a fixed-length tag on the message

properties of MACs

- correctness: determinism
- efficiency: computing MACs should be efficient
- security: EU-CPA (existentially unforgeable under chosen plaintext attack)
  - attacker cannot create a valid tag on a message without the key

HMAC

HMAC

- a MAC used widely in practice offering EU-CPA

HMAC

- a MAC used widely in practice offering EU-CPA
- HMAC(K, M)
  - outputs $H((K' \oplus opad) \| H((K' \oplus ipad) \| M))$
  - not necessary to know, opad and ipad are bytes set by the design of HMAC

HMAC

- a MAC used widely in practice offering EU-CPA
- HMAC(K, M)
  - outputs $H((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, M))$
  - not necessary to know, opad and ipad are bytes set by the design of HMAC
- this is a hash function! same properties

the security of HMAC

the security of HMAC

- does HMAC provide integrity?

the security of HMAC

- does HMAC provide integrity?
  - yes, can’t tamper without detection

the security of HMAC

- does HMAC provide integrity?
  - yes, can’t tamper without detection
- authenticity?

the security of HMAC

- does HMAC provide integrity?
  - yes, can’t tamper without detection
- authenticity?
  - depends: yes if only two people have secret key

the security of HMAC

- does HMAC provide integrity?
  - yes, can’t tamper without detection
- authenticity?
  - depends: yes if only two people have secret key
  - why?

the security of HMAC

- does HMAC provide integrity?
  - yes, can’t tamper without detection
- authenticity?
  - depends: yes if only two people have secret key
  - why?
- confidentiality?

the security of HMAC

- does HMAC provide integrity?
  - yes, can’t tamper without detection
- authenticity?
  - depends: yes if only two people have secret key
  - why?
- confidentiality?
  - no, MACs are deterministic->not IND-CPA secure

the security of HMAC

- does HMAC provide integrity?
  - yes, can’t tamper without detection
- authenticity?
  - depends: yes if only two people have secret key
  - why?
- confidentiality?
  - no, MACs are deterministic->not IND-CPA secure
  - i.e., HMAC is a hash function

authenticated encryption (AE)

authenticated encryption (AE)

- MAC-then-encrypt: Enc($K1$, M || MAC($K2$, M))

authenticated encryption (AE)

- MAC-then-encrypt: $\mathrm{Enc}(K1, M \,||\, \mathrm{MAC}(K2, M))$
- encrypt-then-MAC: $\mathrm{MAC}(K2, \mathrm{Enc}(K1, M))$

authenticated encryption (AE)

- MAC-then-encrypt: $\mathrm{Enc}(K1, M \,||\, \mathrm{MAC}(K2, M))$
- encrypt-then-MAC: $\mathrm{MAC}(K2, \mathrm{Enc}(K1, M))$
- which is better?

authenticated encryption (AE)

- MAC-then-encrypt: Enc($K1$, M || MAC($K2$, M))
- encrypt-then-MAC: MAC($K2$, Enc($K1$, M))
- which is better?
  - both are technically IND-CPA and EU-CPA

authenticated encryption (AE)

- MAC-then-encrypt: Enc($K1$, M || MAC($K2$, M))
- encrypt-then-MAC: MAC($K2$, Enc($K1$, M))
- which is better?
  - both are technically IND-CPA and EU-CPA
  - but MAC-then-encrypt requires decryption before tag verification

authenticated encryption (AE)

- MAC-then-encrypt: $\mathrm{Enc}(K1, M \,||\, \mathrm{MAC}(K2, M))$
- encrypt-then-MAC: $\mathrm{MAC}(K2, \mathrm{Enc}(K1, M))$
- which is better?
  - both are technically IND-CPA and EU-CPA
  - but MAC-then-encrypt requires decryption before tag verification
  - **always use encrypt-then-MAC**

authenticated encryption (AE)

- MAC-then-encrypt: Enc($K1$, M || MAC($K2$, M))
- encrypt-then-MAC: MAC($K2$, Enc($K1$, M))
- which is better?
  - both are technically IND-CPA and EU-CPA
  - but MAC-then-encrypt requires decryption before tag verification
  - **always use encrypt-then-MAC**
    - more robust to mistakes

diffie-hellman

key exchange

motivation for diffie-hellman

motivation for diffie-hellman

- lots of our encryption/MAC schemes require a shared secret—a key

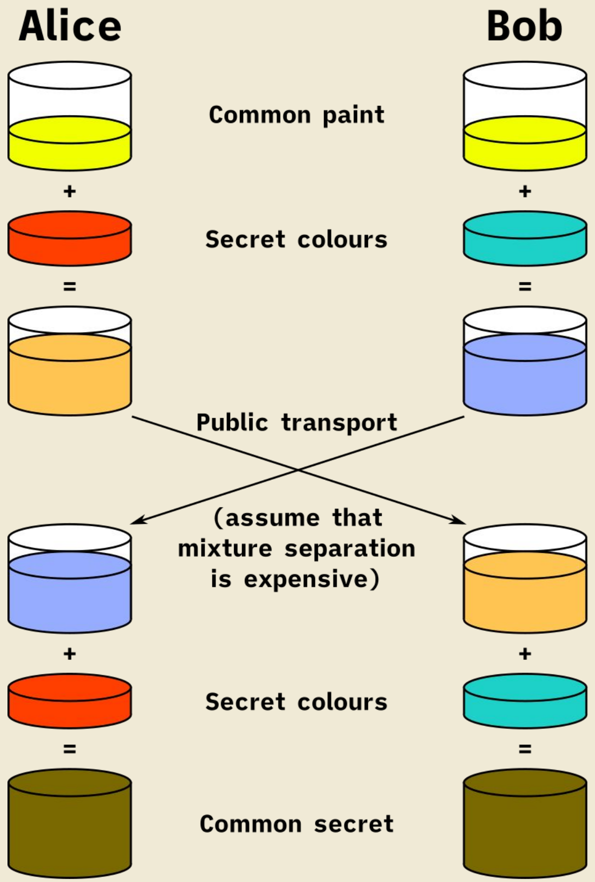
motivation for diffie-hellman

- lots of our encryption/MAC schemes require a shared secret—a key
- what if you don’t have one?

motivation for diffie-hellman

- lots of our encryption/MAC schemes require a shared secret—a key
- what if you don’t have one?
    - how do we exchange keys/secrets securely?

color sharing—diffie-hellman



discrete log problem

- in the color sharing example, why couldn't Eve get the key? separating mixed colors is hard.

discrete log problem

- in the color sharing example, why couldn’t Eve get the key? separating mixed colors is hard.
- we need a mathematical equivalent of this

discrete log problem

- in the color sharing example, why couldn’t Eve get the key? separating mixed colors is hard.
- we need a mathematical equivalent of this
- given g, p and g^a mod p for unknown ‘a’, it is computationally hard to find ‘a’

diffie-hellman key exchange (lecture)

Public: g, p

Alice

Eve

Bob

Secret key → Generate a

Generate b

Public key → Calculate g^a mod p

g^a g^b

Calculate g^b mod p

Receive g^b mod p

Receive g^a mod p

Calculate $(g^b)^a$ mod p

Calculate $(g^a)^b$ mod p

$a, g^a, g^b \Rightarrow g^{ab}$

$g^a, g^b \not\Rightarrow g^{ab}$

$b, g^a, g^b \Rightarrow g^{ab}$

Shared symmetric key is g^{ab}

diffie-hellman’s legitimacy

diffie-hellman's legitimacy

- <u>discrete log problem:</u> given g, p, g^a mod p for random 'a', it is computationally hard to find 'a'

diffie-hellman's legitimacy

- <u>discrete log problem:</u> given g, p, g^a mod p for random '*a*', it is computationally hard to find '*a*'
- <u>ephemeral:</u> a, b, and shared key discarded when done

diffie-hellman's legitimacy

- <u>discrete log problem:</u> given *g*, *p*, g^a mod *p* for random '*a*', it is computationally hard to find '*a*'
- <u>ephemeral:</u> a, b, and shared key discarded when done
- <u>forward secrecy:</u> even if a future secret is stolen, old messages cannot be decrypted—*a*, *b*, and the shared secret *K* were never recorded

why can't eve reconstruct g^{ab}?

Public: g, p


Alice

Mallory

Bob


Secret key

Public key

Alice	Mallory	Bob
Generate a		Generate b
Calculate g^a mod p	g^a $\quad$ g^b	Calculate g^b mod p
Receive g^b mod p		Receive g^a mod p
Calculate $(g^b)^a$ mod p		Calculate $(g^a)^b$ mod p
$a, g^a, g^b \Rightarrow g^{ab}$	$g^a, g^b \not\Rightarrow g^{ab}$	$b, g^a, g^b \Rightarrow g^{ab}$

Shared symmetric key is g^{ab}

why can't eve reconstruct g^{ab}?

Public: g, p

Alice

Eve

Bob

Alice	Eve	Bob
Generate a		Generate b
Calculate g^a mod p	g^a g^b	Calculate g^b mod p
Receive g^b mod p		Receive g^a mod p
Calculate $(g^b)^a$ mod p		Calculate $(g^a)^b$ mod p
$a, g^a, g^b \Rightarrow g^{ab}$	$g^a, g^b \not\Rightarrow g^{ab}$	$b, g^a, g^b \Rightarrow g^{ab}$

Secret key

Public key

Shared symmetric key is g^{ab}

how can **mallory** read alice and bob's communications?

Public: g, p

Alice

Secret key

Public key

Generate a

Calculate g^a mod p

Receive g^m mod p

Calculate $(g^m)^a$ mod p

$a, g^a, g^m \Rightarrow g^{am}$

Mallory

g^a

Generate m

g^m

Calculate g^m mod p

Compute $(g^a)^m = g^{am}$

Compute $(g^b)^m = g^{bm}$

Bob

Generate b

Calculate g^b mod p

g^b

g^m

Receive g^m mod p

Calculate $(g^m)^b$ mod p

$b, g^m, g^b \Rightarrow g^{mb}$

Shared symmetric key is g^{am} and g^{bm}

worksheet

(on 161 website)

feedback

bit.ly/extended-feedback

slides: **bit.ly/cs161-disc**