CS 88: Security and Privacy

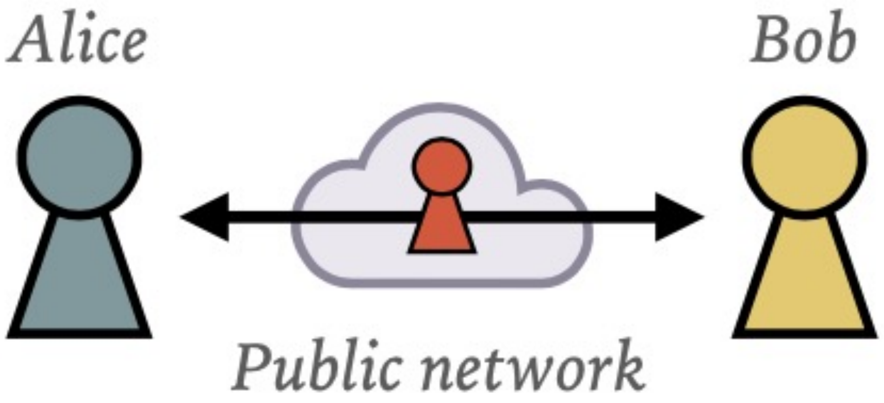
16: MACs and PKI

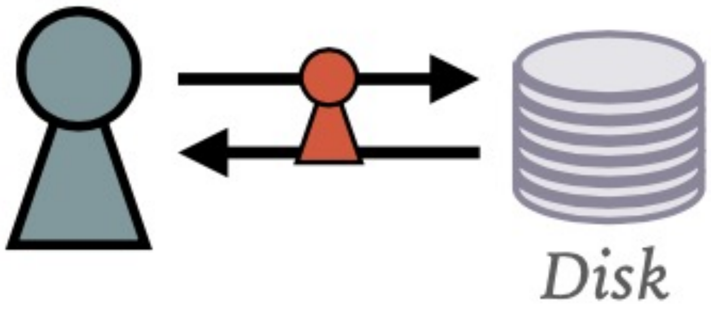
03-26-2024

slides courtesy Christo Wilson, Vitaly Shmatikov

SWARTHMORE COLLEGE

Symmetric Key Cryptography





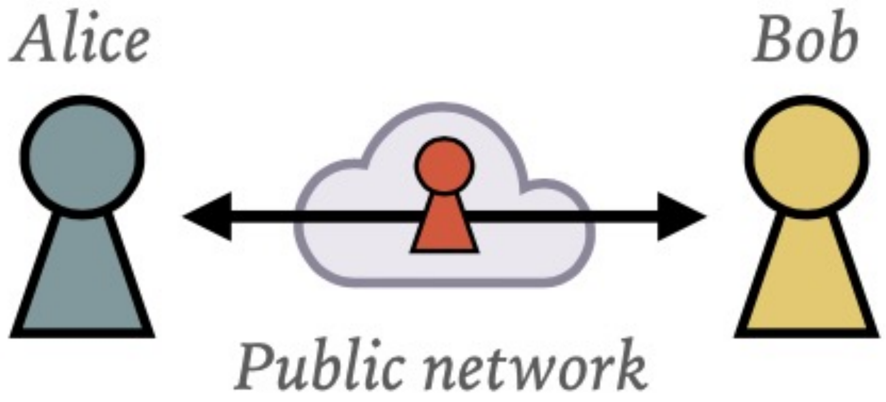
Confidentiality	Keep others from reading Alice's messages/data
Integrity	Keep others from undetectably tampering with Alice's messages/data
Authenticity	Keep others from undetectably impersonating Alice (keep her to her word too!)

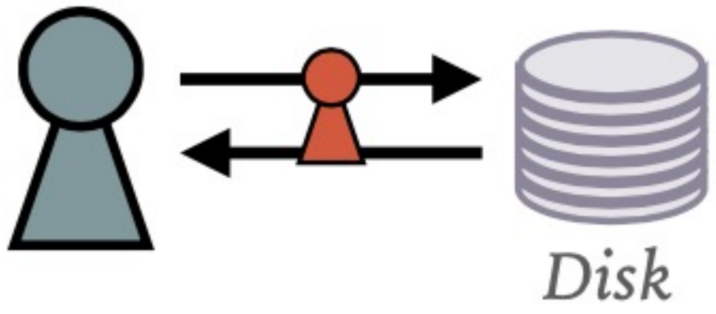
Block Ciphers

Limitations?

- what if Eve modifies the packet in transit?
- How do we share keys?

Scenarios and Goals





Confidentiality — Keep others from reading Alice's messages/data

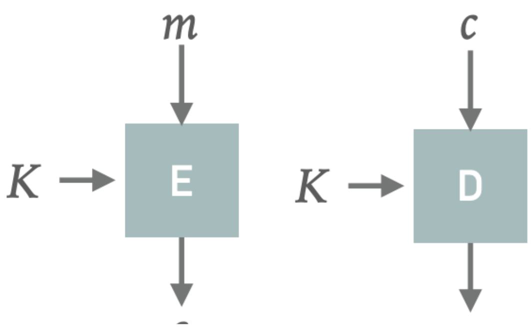
Integrity — Keep others from undetectably tampering with Alice's messages/data

Message Authentication Codes (MACs)

Authenticity — Keep others from undetectably impersonating Alice (keep her to her word too!)

BLACKBOX #2:
MESSAGE AUTHENTICATION CODE (MAC)

Symmetric Key Cryptography



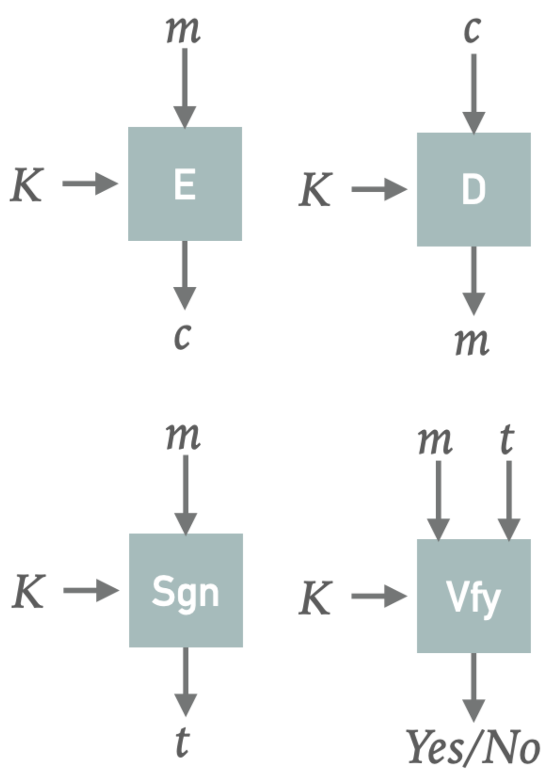
CONFIDENTIALITY

Block ciphers

Deterministic $\implies$ *use IVs*

Fixed block size $\implies$ *use encryption "modes"*

Could we simply use symmetric key cryptography (i.e. block ciphers) to achieve integrity?



CONFIDENTIALITY

Block ciphers

Deterministic $\Longrightarrow$ *use IVs*

Fixed block size $\Longrightarrow$ *use encryption "modes"*

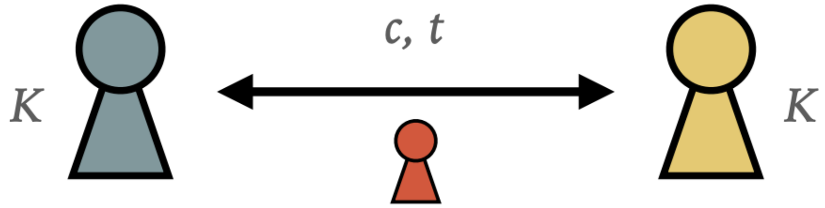
INTEGRITY

Message Authentication Codes (MACs)

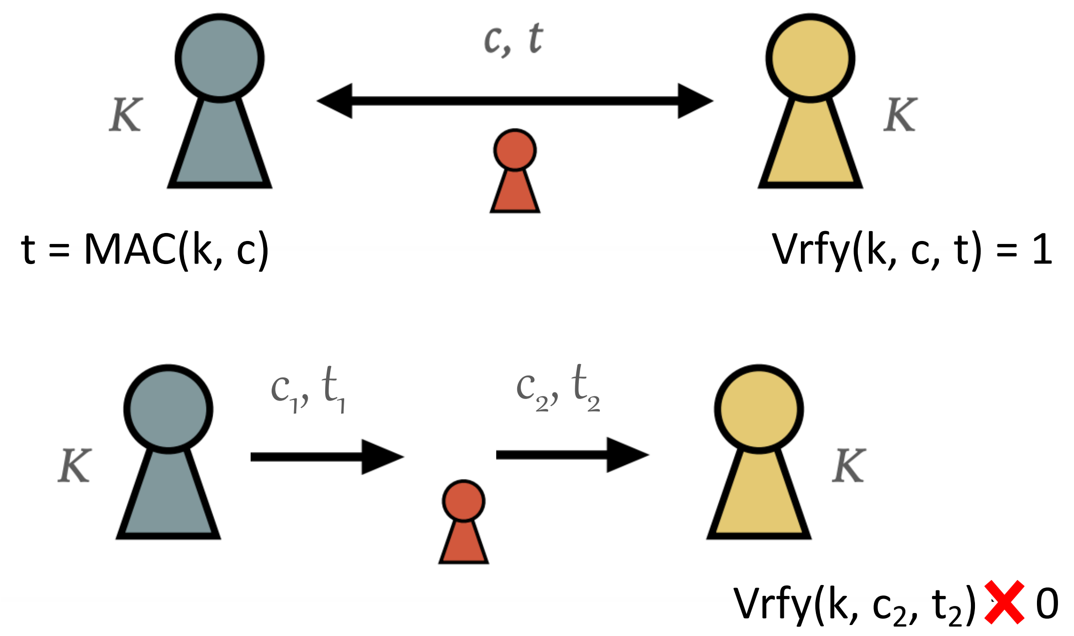
Send (message, tag) pairs

Verify that they match

A. Yes
B. No
C. Maybe
D. Under some circumstances



Confidentiality vs. Integrity



Ensuring that a received ciphertext **originated** from the intended party, and the ciphertext was **not modified.**

Even if an attacker controls the channel!

Message Authentication Codes

A message authentication code is defined by three PPT algorithms (Gen, Mac, Vrfy):

– Gen: takes as input an n bit string; outputs k. (Assume $|k| \geq n$.)
– Mac: takes as input key k and message $m \in \{0,1\}^*$; outputs tag t t := Mac(k, m)
– Vrfy: takes key k, message m, and tag t as input; outputs 1 (“accept”) or 0 (“reject”)

For all m and all k output by Gen, Vrfy(K, m, Mac(k, m)) = 1

Message Authentication Codes

- Sign: takes a key and a message and outputs a “tag”
  - Sgn(k,m) = t

- Verify: takes a key, a message, and a tag, and outputs Y/N
  - Vfy(k,m,t) = {Y,N}

- Correctness:
  - Vfy(k, m, Sgn(k, m)) = Y (or 1)

General adversarial goals

- Total Break: Adversary is able to find the secret key for signing and forge any signature of any message
- Selective forgery: Adversary is able to create valid signatures on a message chosen by someone else, with a significant probability.
- Existential Forgery: Adversary can create a pair of (message, signature) such that the signature of the message is valid.

- Ciphertext only Attack: Adversary knows only the verification function
- Known Plaintext Attack: Adversary knows a list of messages previously signed by Alice
- Chosen Plaintext Attack: Adversary can choose what messages they want Alice to sign, and knows both the message and the corresponding signature

Attacker Goal: Existential Forgery

- A MAC is secure if an attacker cannot demonstrate an existential forgery despite being able to perform a chosen plaintext attack:

- Chose plaintext:
  - Attacker gets to choose m1, m2, m3, …
  - And in return gets a properly computed t1, t2, t3, …

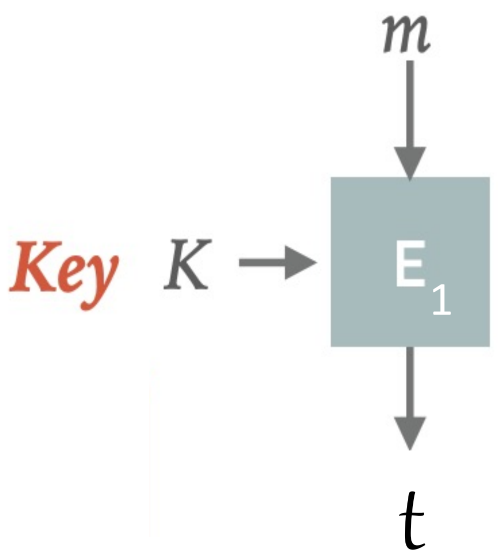
- Existential forgery:
  - Construct a new (m,t) pair such that Vfy(k, m, t) = Y

Attacker Goal: Existential Forgery

- A MAC is secure if an attacker cannot demonstrate an existential forgery despite being able to perform a chosen plaintext attack:
- Chose plaintext:
  - Attacker gets to choose m1, m2, m3, …
  - And in return gets a properly computed t1, t2, t3, …
- Existential forgery:
  - Construct a new (m,t) pair such that Vfy(k, m, t) = Y
- Let MAC be a pseudorandom function!

Block Ciphers as fixed length MACs

ENCRYPTION



Encryption Function: E: $\{0, 1\}^k$ x $\{0, 1\}^n$ -> $\{0, 1\}^n$

Fix the key K, then, E_k: $\{0, 1\}^n$ -> $\{0, 1\}^n$

- plaintext size: n
- tag size:n

E_k: permutation on n-bit strings.

- invertible (bijective function) given the key

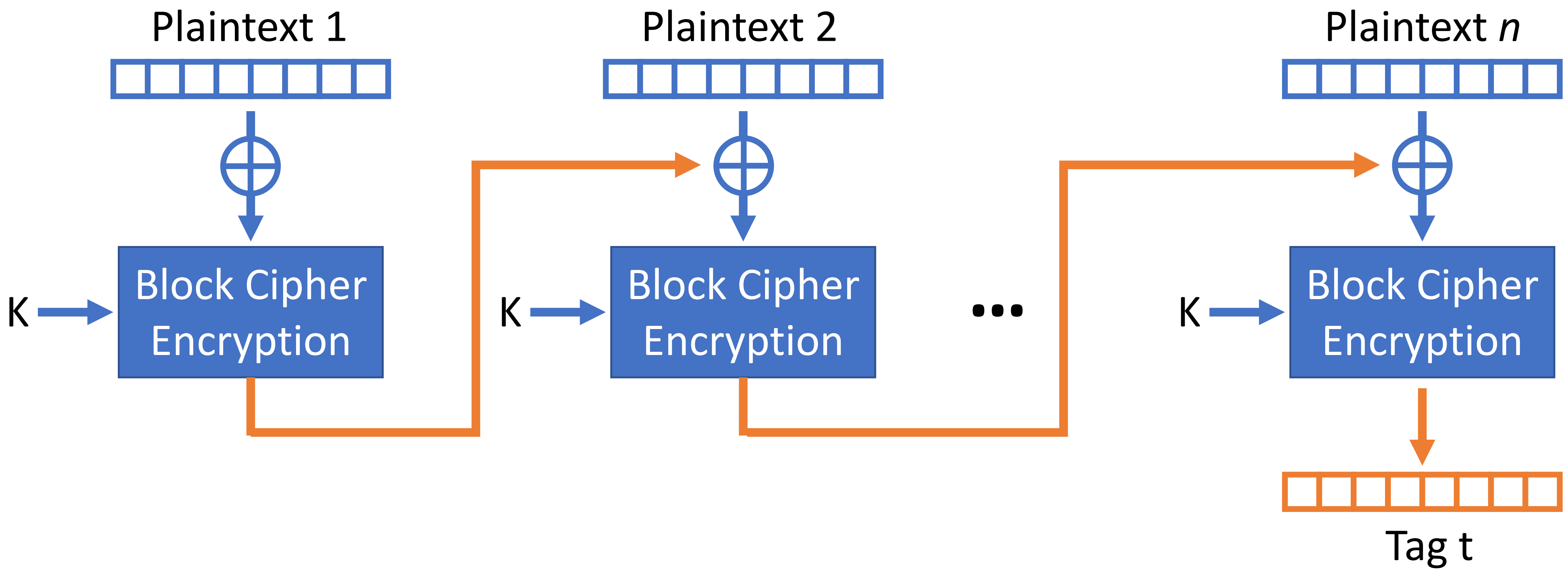
Once the key is fixed: MAC(k,m) is indistinguishable from a function chosen uniformly at random from all possible functions between block-sized binary strings.

Block Ciphers as fixed length MACs

- We can construct a secure MAC for short, fixed-length messages based on any block cipher
- But we want to extend this to a secure MAC for arbitrary-length messages.
  - What can we do?
  - CBC-MAC!

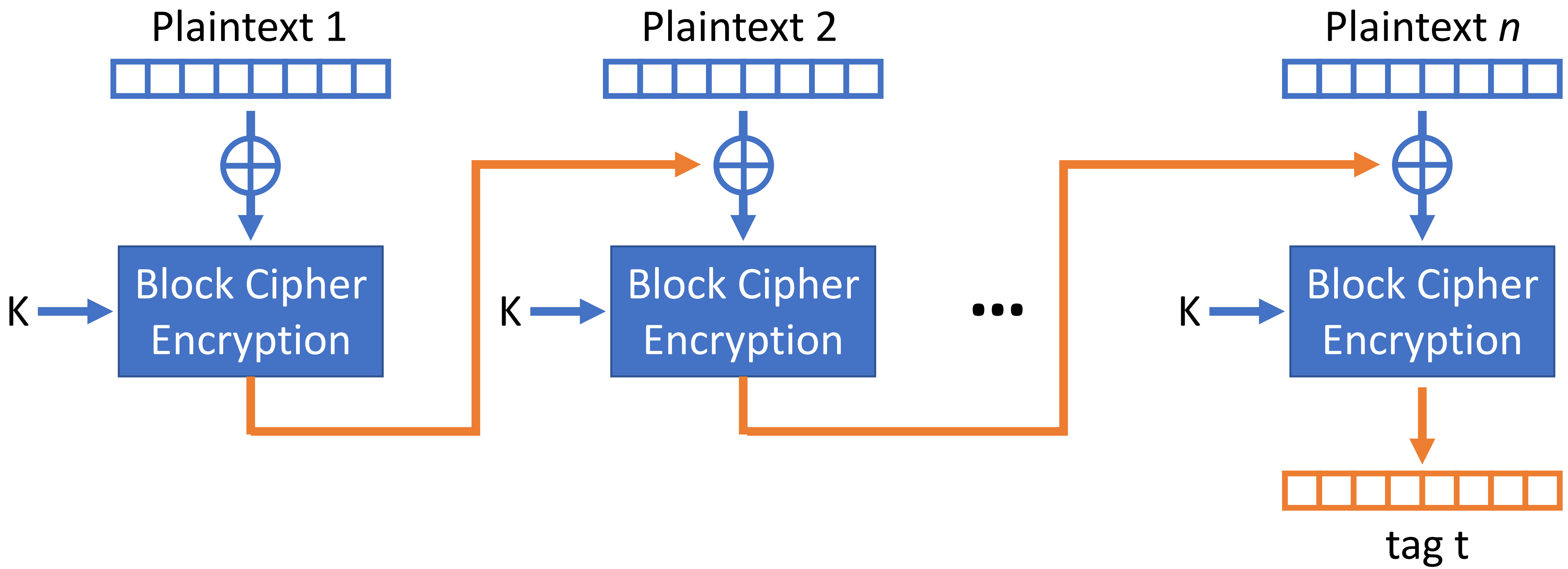
CBC MAC

- What is one important difference you observe compared to CBC-Mode encryption?



CBC MAC

- CBC-MAC is deterministic (no IV)
- In CBC-MAC, only the final value is output (tag t) – Verification is done by re-computing the result



BLACKBOX #3:
HASH FUNCTIONS

Hash Function Properties

- Very fast to compute
- Takes arbitrarily-sized inputs, returns fixed-sized output
- Pre-image resistant:

  Given $H(m)$, hard to determine m
- Collision resistant

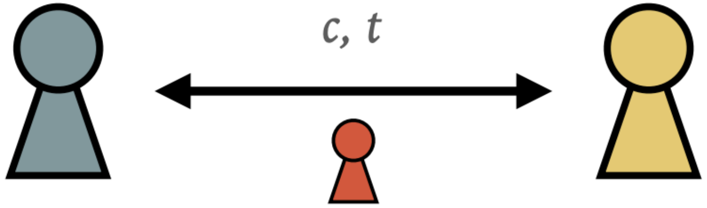
  Given m and $H(m)$, hard to find $m' \neq m$ s.t. $H(m) = H(m')$

Good hash functions: SHA family (SHA-256, SHA-512, …)

Authenticated Encryption: Secrecy + Integrity

We have seen how we can achieve two independent goals: encryption and authentication. How about putting them together?

k1, k2



k1, k2

```
m
Enc(k1, m) = c
Mac(k2, m) = t
```

```
Dec(k1, c) = m
Verify(k2, m, t) = 1?
```

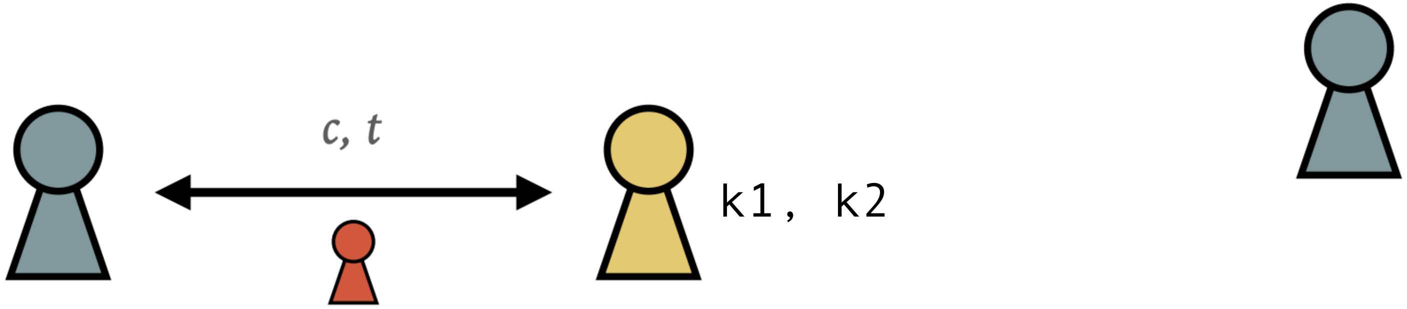
Encrypt and Authenticate: Is it secure?

A. Yes, encryption is randomized with proper K, IV
B. No the tag might leak information
C. No the MAC is deterministic

Encrypt then authenticate

We have seen how we can achieve two independent goals: encryption and authentication.
How about putting them together?

k1, k2



k1, k2

```
m
Enc(k1, m) = c
Mac(k2, c) = t
```

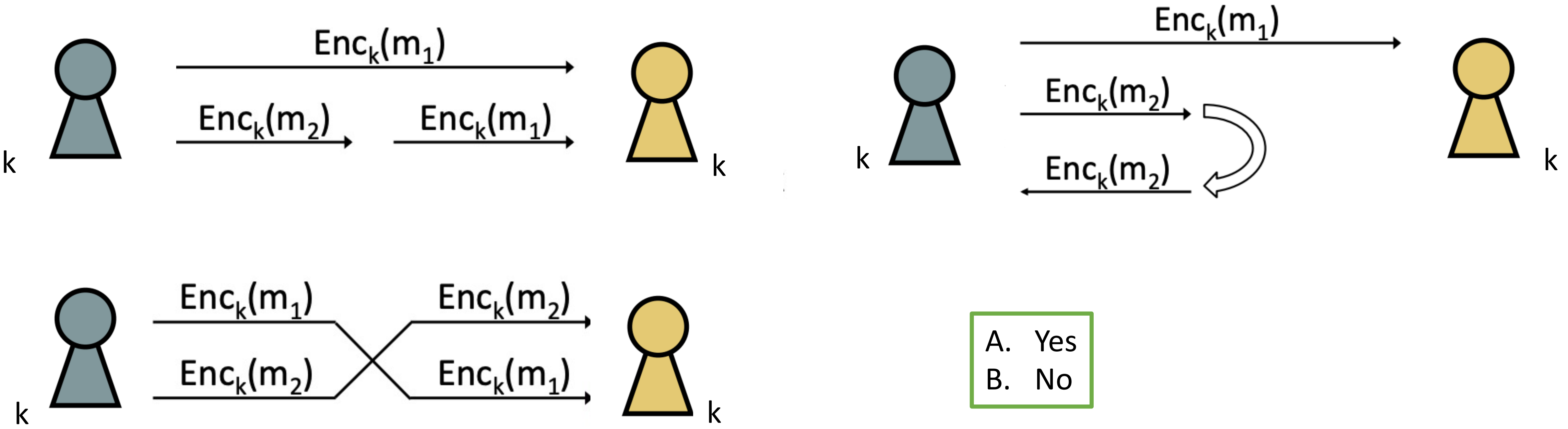
```
Verify(k2, c, t) = 1?
Dec(k1, c) = m
```

Encrypt then Authenticate: Is it secure?

A. Yes, encryption is randomized with proper K, IV
B. No the tag might leak information
C. No the MAC is deterministic

Secure Sessions: Consider parties who wish to communicate securely over the course of a session using authenticated encryption. Are they immune to the following attacks?

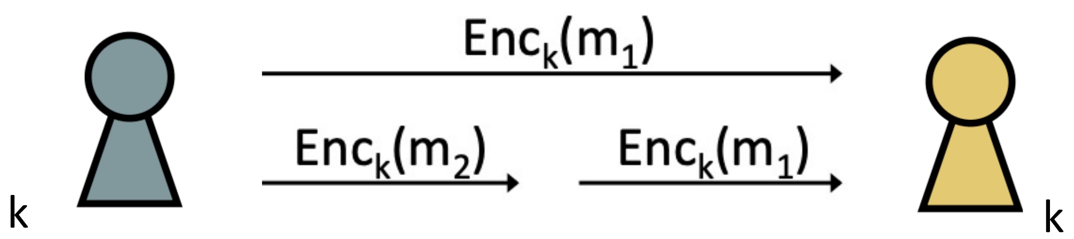
- Securely = secrecy and integrity
- Session = period of time over which parties are willing to maintain state.

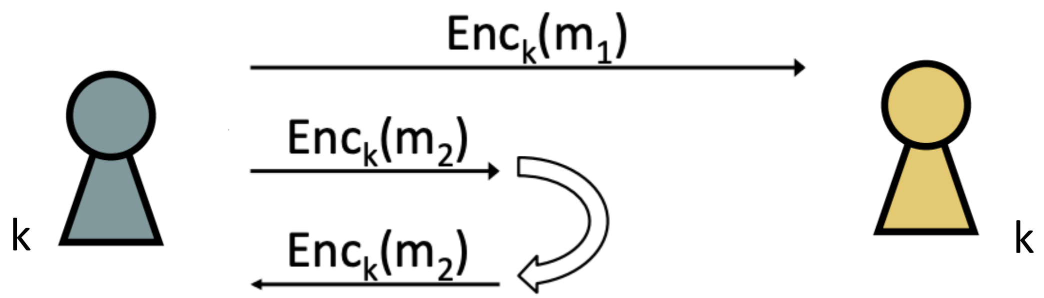


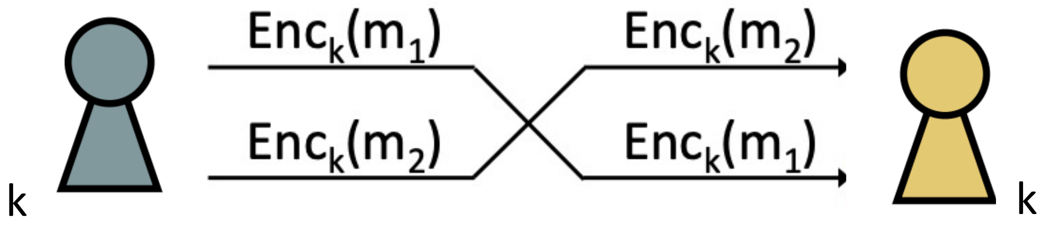
A. Yes
B. No

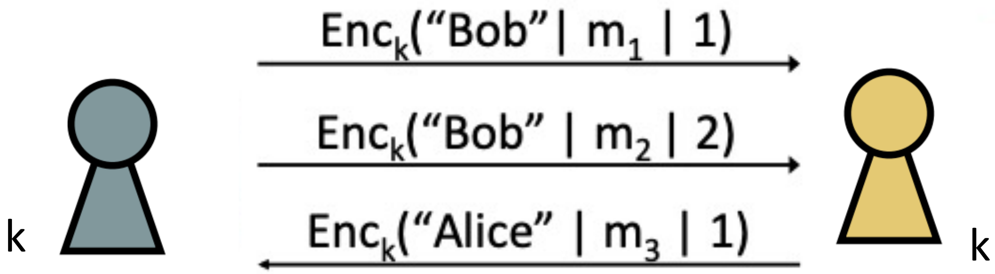
Secure Sessions: Consider parties who wish to communicate securely over the course of a session using authenticated encryption. Are they immune to the following attacks?

- Securely = secrecy and integrity
- Session = period of time over which parties are willing to maintain state.

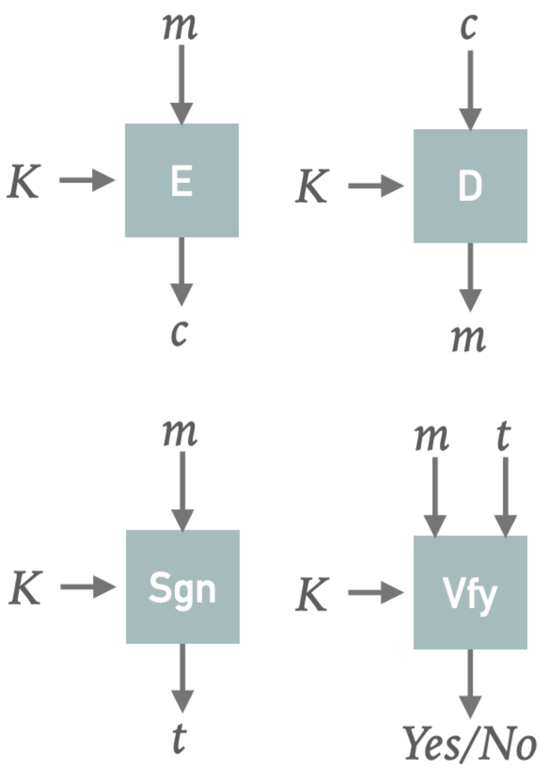








Symmetric Key Cryptography



CONFIDENTIALITY

Block ciphers

Deterministic $\implies$ use IVs

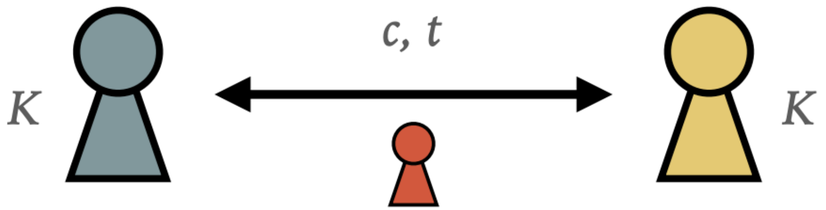
Fixed block size $\implies$ use encryption "modes"

INTEGRITY

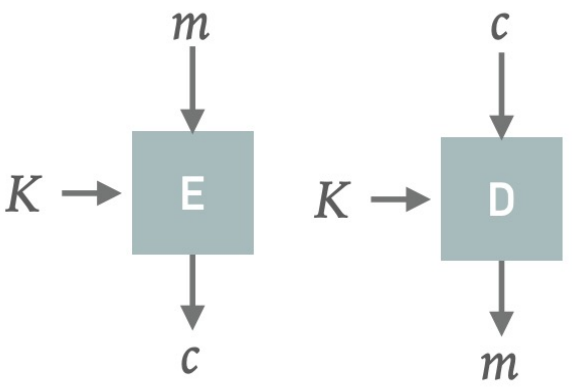
Message Authentication Codes (MACs)

Send (message, tag) pairs

Verify that they match



Symmetric Key Cryptography

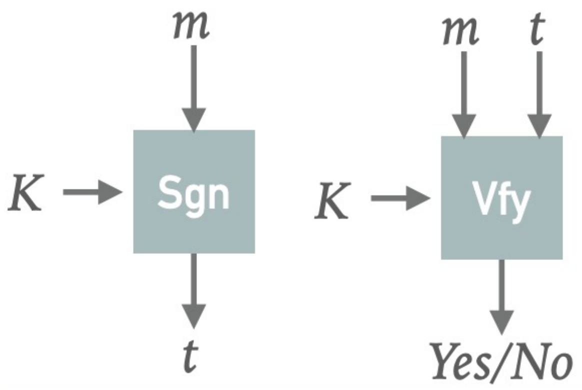


CONFIDENTIALITY

Block ciphers

Deterministic ⟹ use IVs

Fixed block size ⟹ use encryption “modes”



INTEGRITY

Message Authentication Codes (MACs)

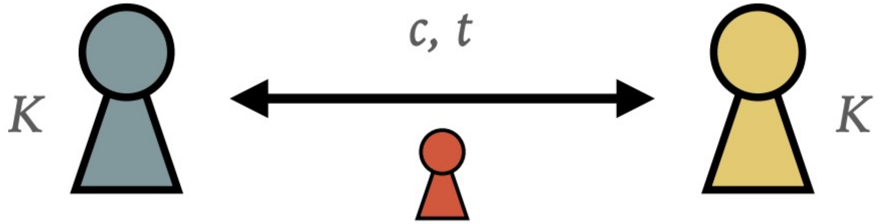
Send (message, tag) pairs

Verify that they match

Next

How do we establish K?

How do we know with whom we are communicating?



BLACKBOX #4:
DIFFIE HELLMAN KEY ESTABLISHMENT

Asymmetric/Public-key Cryptography

- main insight: separate keys for different functions
- Keys come in pairs, and are related to each other by a specific algorithm.
  - Public key (PK): used to encrypt or verify signatures
  - Private key (SK): used to decrypt and sign
- Encryption and decryption are inverse operations
- Secrecy: ciphertext reveals nothing about the plaintext
  - computationally hard to decrypt in polynomial time without key

Diffie-Helman Key Exchange

$x \bmod N$

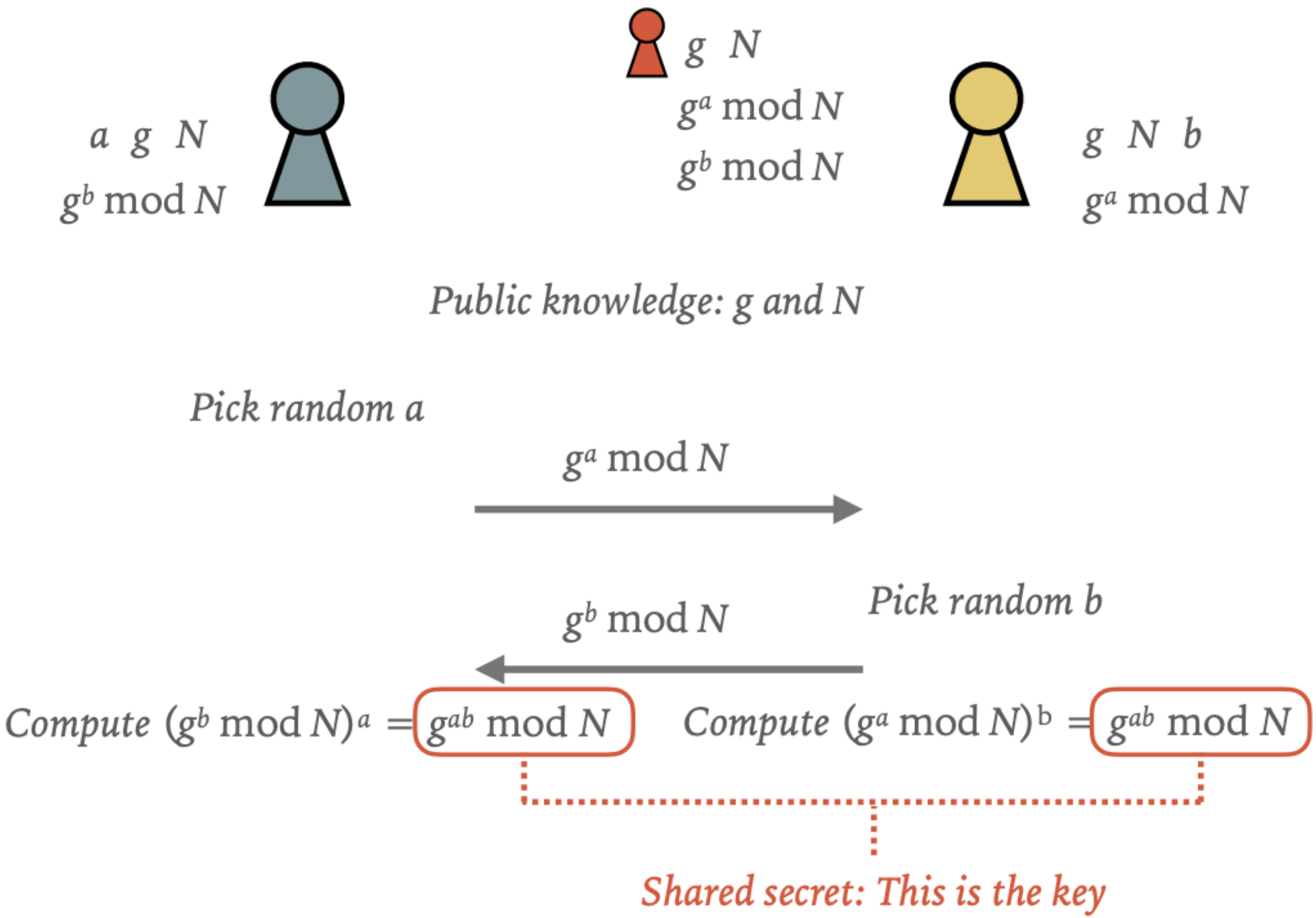
g is a generator of mod N if
$\{1, 2, \ldots, N\text{-}1\} = \{g^0 \bmod N, g^1 \bmod N, \ldots, g^{N-2} \bmod N\}$

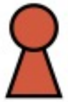
$N=5, g=3$
$3^0 \bmod 5 = 1$ $3^1 \bmod 5 = 3$ $3^2 \bmod 5 = 4$ $3^3 \bmod 5 = 2$

Given x and g, it is efficient to compute
$g^x \bmod N$

Given g and g^x, it is efficient to compute x
(simply take $\log_g g^x$)

Given g and g^x mod N it is *infeasible* to compute x
Discrete log problem


g N
$g^a \bmod N$
$g^b \bmod N$
a g N
$g^b \bmod N$
g N b
$g^a \bmod N$
Public knowledge: g and N
Pick random a
$g^a \bmod N$
Pick random b
$g^b \bmod N$
Compute $(g^b \bmod N)^a = \boxed{g^{ab} \bmod N}$
Compute $(g^a \bmod N)^b = \boxed{g^{ab} \bmod N}$
Shared secret: This is the key

 $g \quad N$

$g^a \bmod N$

$g^b \bmod N$

$\boxed{g^{ab} \bmod N}$

Note that just multiplying g^a and g^b won't suffice:

$$g^a \bmod N * g^b \bmod N \quad = g^{a+b} \bmod N$$

Key property:

An *eavesdropper* cannot infer the shared secret (g^{ab}).

But what about *active intermediaries*?

 $g \quad N$

$g^a \bmod N$

$g^b \bmod N$

$\boxed{g^{ab} \bmod N}$

Given g and g^x mod N it is *infeasible* to compute x
Discrete log problem

Note that just multiplying g^a and g^b won't suffice:

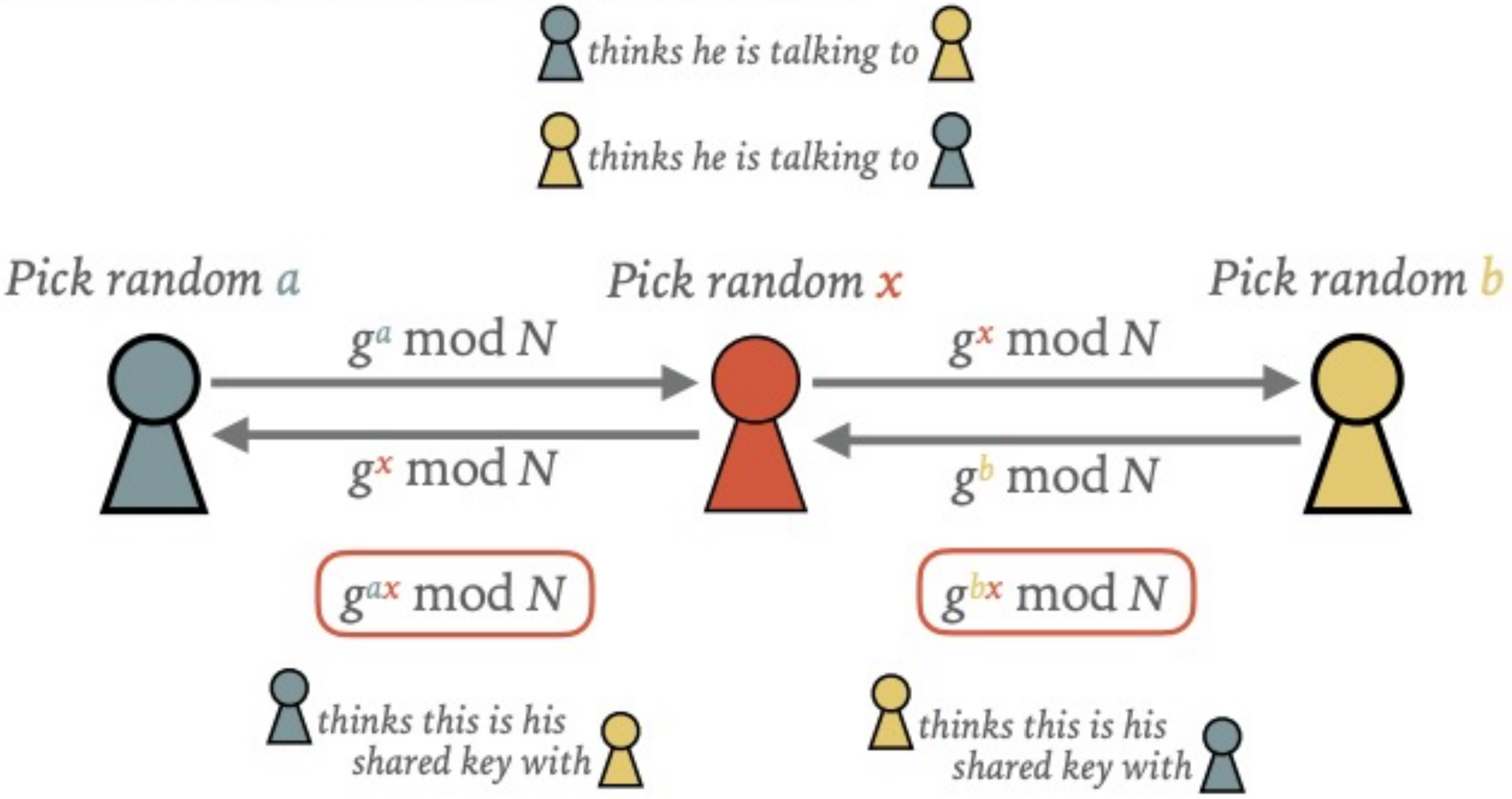
$$g^a \bmod N * g^b \bmod N \quad = g^{a+b} \bmod N$$

Key property:

An *eavesdropper* cannot infer the shared secret (g^{ab}).

But what about *active intermediaries*?

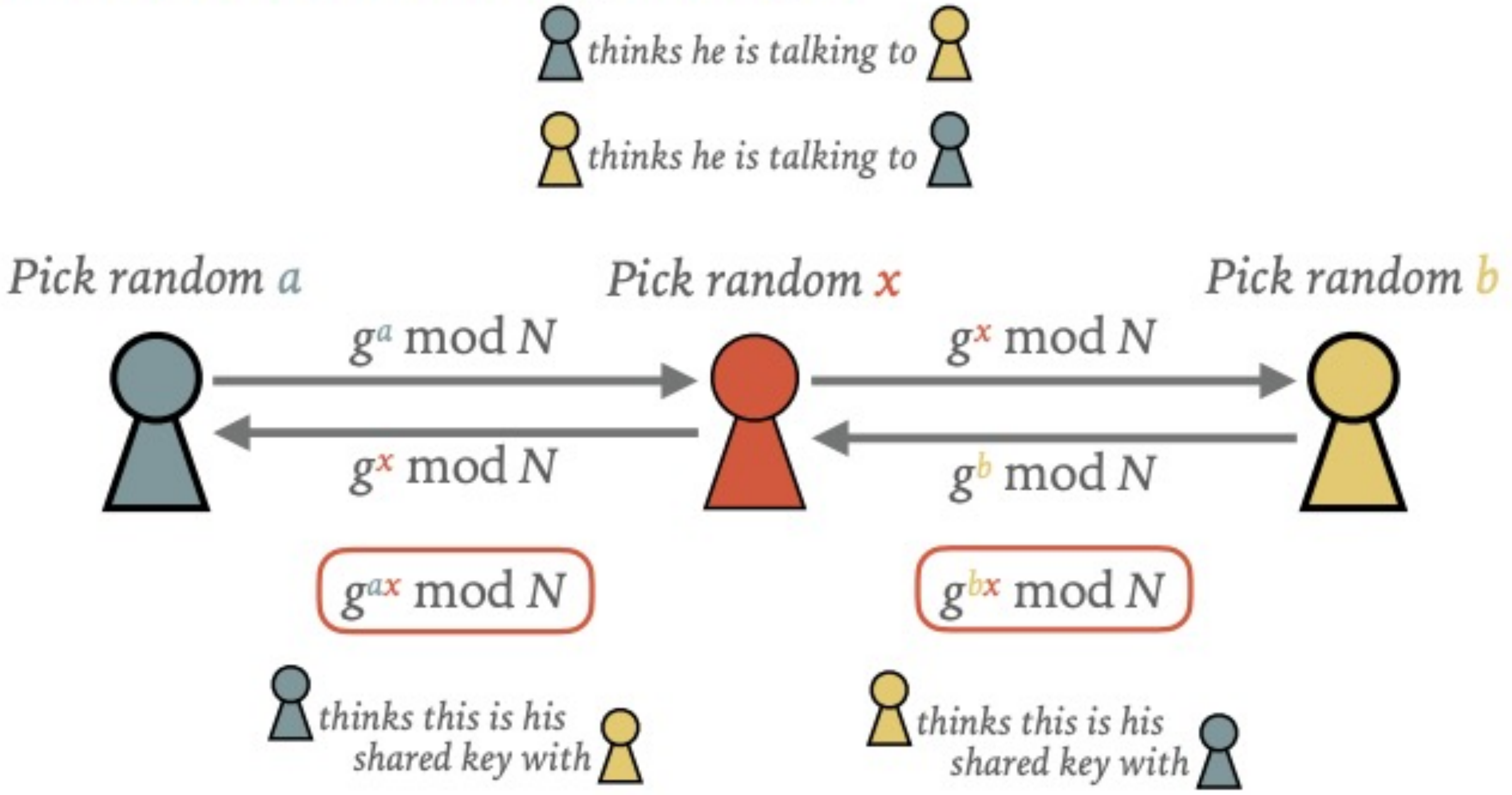
The attacker can interpose between the two communicating parties and insert, delete, and modify messages.



The attacker can now eavesdrop on the conversation.

Key property: Diffie-Hellman is *not* resilient to a MITM attack

The attacker can interpose between the two communicating parties and insert, delete, and modify messages.



The attacker can now eavesdrop on the conversation.

Key property: Diffie-Hellman is *not* resilient to a MITM attack

Fix: Need to authenticate messages

Computational complexity for integer problems

- Integer multiplication is efficient to compute
- There is no known polynomial-time algorithm for general purpose factoring.
- Efficient factoring algorithms for many types of integers. *Easy to find small factors of random integers.*
- Modular exponentiation is efficient to compute
- Modular inverses are efficient to compute

Textbook RSA Encryption

Public Key pk

N = pq *modulus*

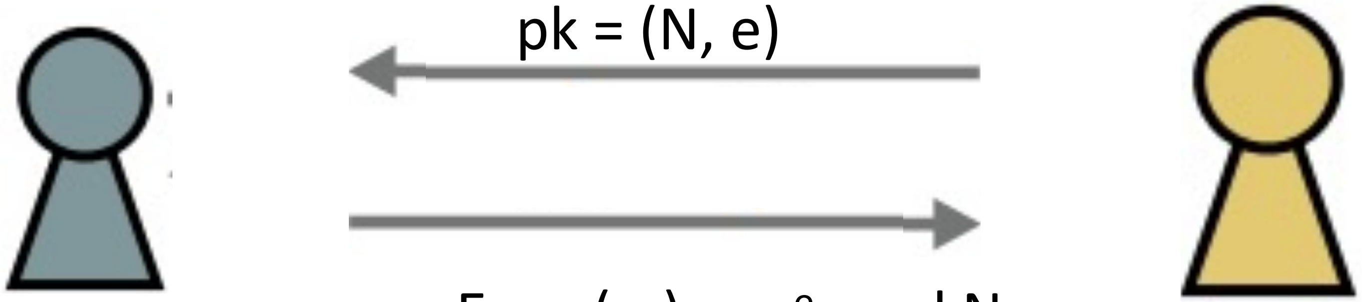
e *encryption exponent*

Secret key sk

p, q *primes*

d *decryption exponent*

$d = e^{-1} \bmod (p-1)(q-1) = e^{-1} \bmod \Phi(N)$



pk = (N, e)

$c = \mathrm{Enc}_{PK}(m) = m^e \bmod N$

$d = \mathrm{Dec}_{SK}(c) = c^d \bmod N$

RSA Security

- Best algorithm to break RSA: Factor N and compute d
- Factoring is not efficient in general
- Current key size recommendations: N >= 2048 bits
- *Do not implement this yourself. Factoring is hard only for some integers, and textbook RSA is insecure.*