

CS 88: Security and Privacy

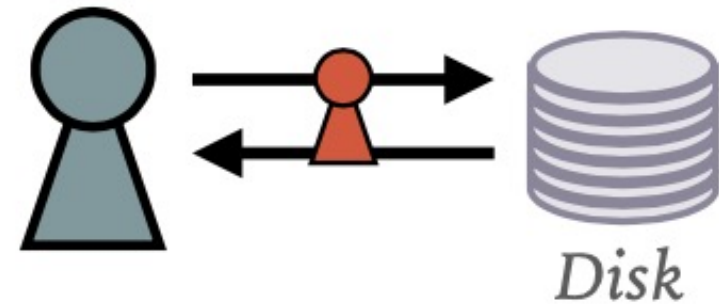
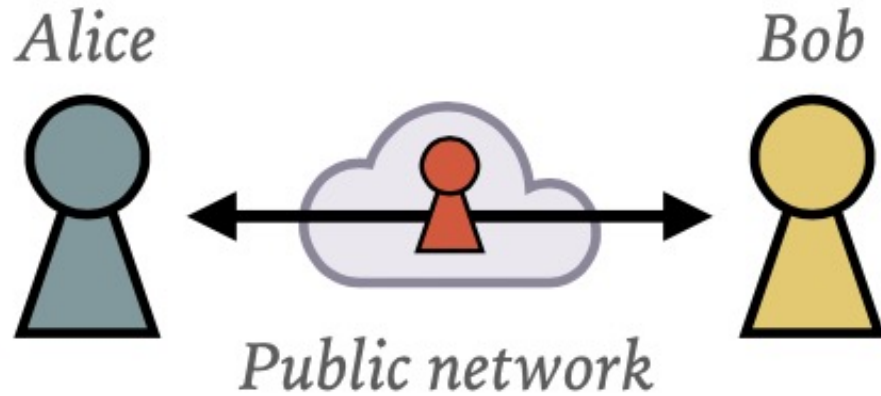
15: MACs, Hash Functions, Diffie-Hellman Key Exchange

10-27-2022

slides adapted from Dave Levine, Jonathan Katz, Kevin Du



Symmetric Key Cryptography



Confidentiality

Keep others from reading Alice's messages/data

Integrity

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

Authenticity

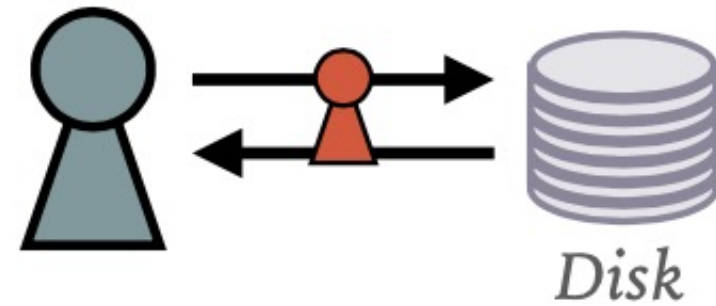
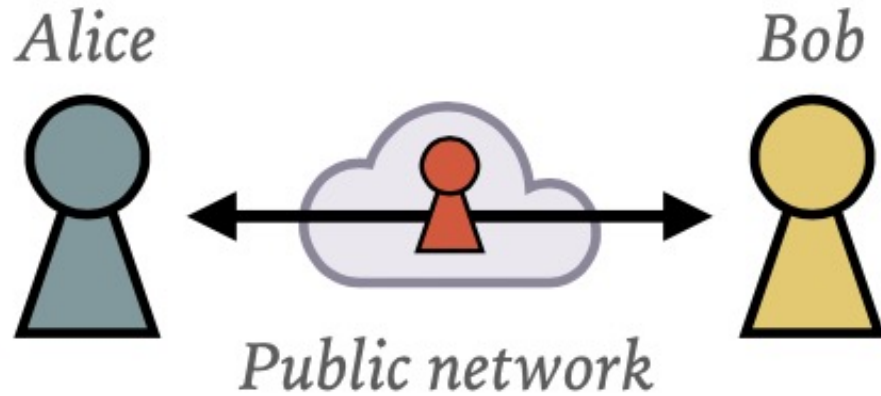
Keep others from undetectably impersonating Alice (keep her to her word too!)

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

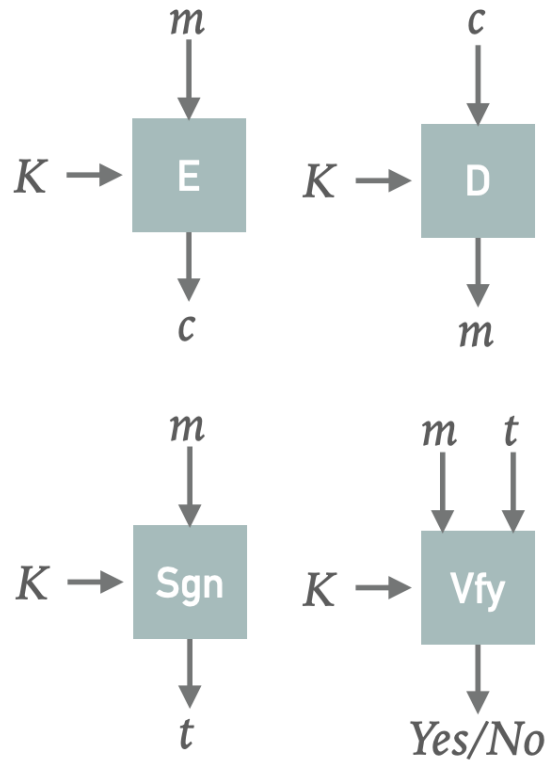
Authenticity

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

Message Authentication Codes (MACs)

BLACKBOX #2:
MESSAGE AUTHENTICATION CODE (MAC)

Symmetric Key Cryptography



CONFIDENTIALITY

Block ciphers

Deterministic \Rightarrow use IVs

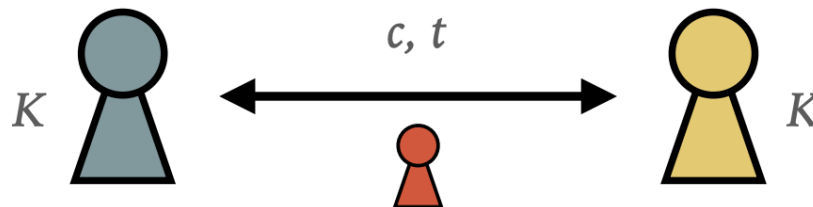
Fixed block size \Rightarrow use encryption "modes"

INTEGRITY

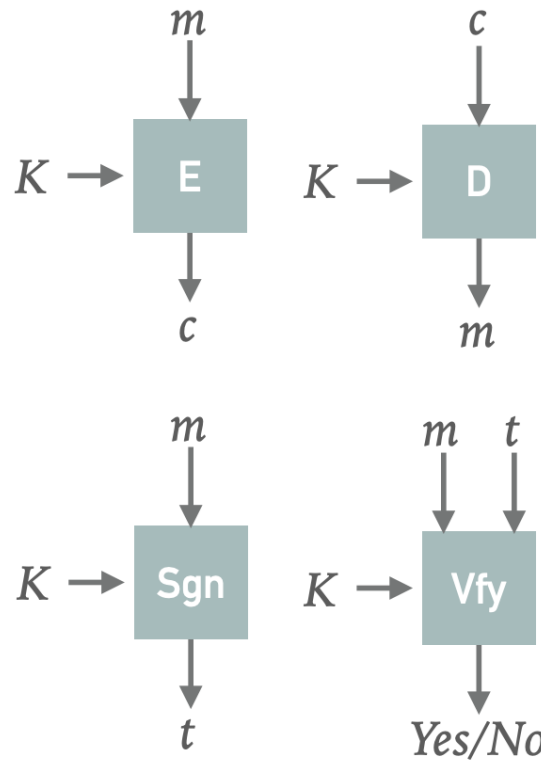
Message Authentication Codes (MACs)

Send (message, tag) pairs

Verify that they match



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



CONFIDENTIALITY

Block ciphers

Deterministic \Rightarrow use IVs

Fixed block size \Rightarrow use encryption "modes"

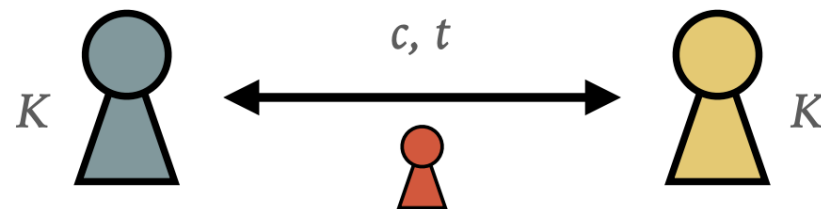
INTEGRITY

Message Authentication Codes (MACs)

Send (message, tag) pairs

Verify that they match

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



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 m_1, m_2, m_3, \dots
 - And in return gets a properly computed t_1, t_2, t_3, \dots
- Existential forgery:
 - Construct a new (m,t) pair such that $\forall y(k, m, t) = Y$

BLACKBOX #3: **HASH FUNCTIONS**

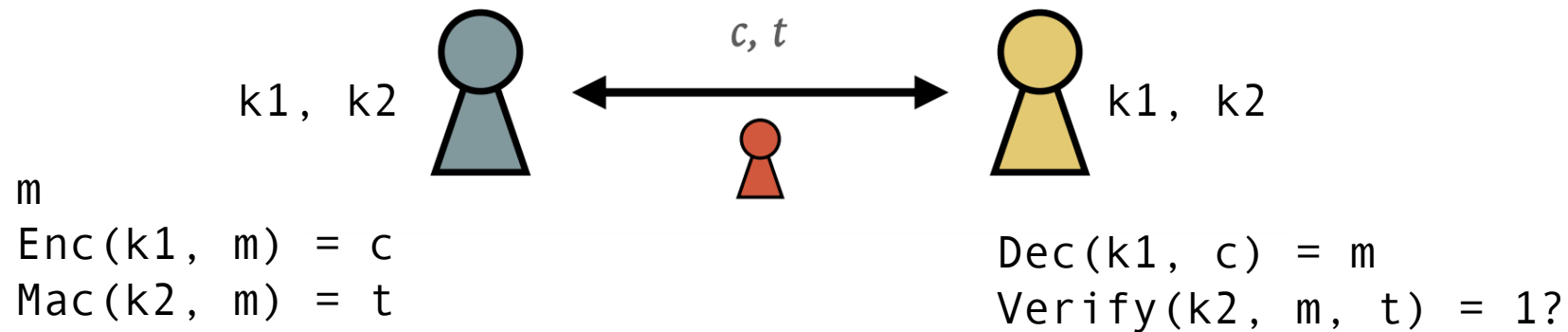
Hash Function Properties

- 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?

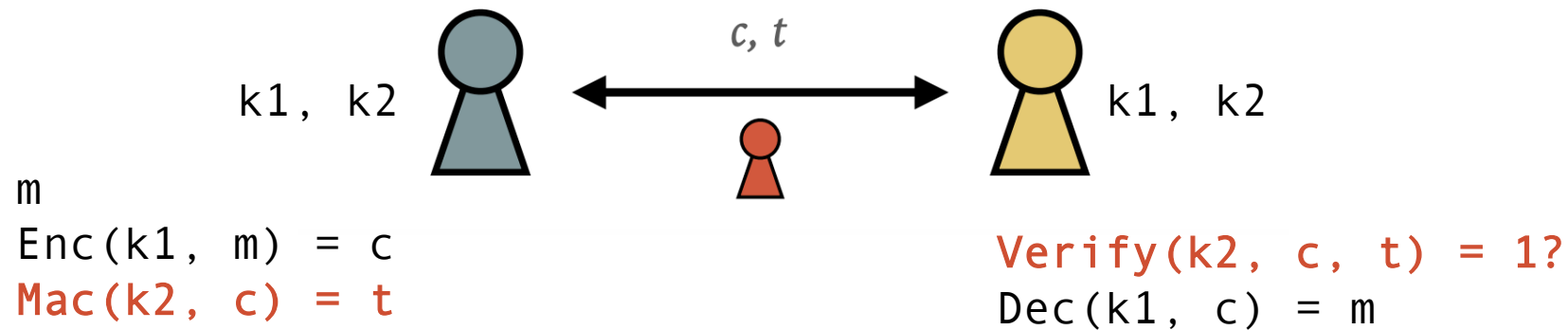


Encrypt and Authenticate: Is it secure?

- 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?

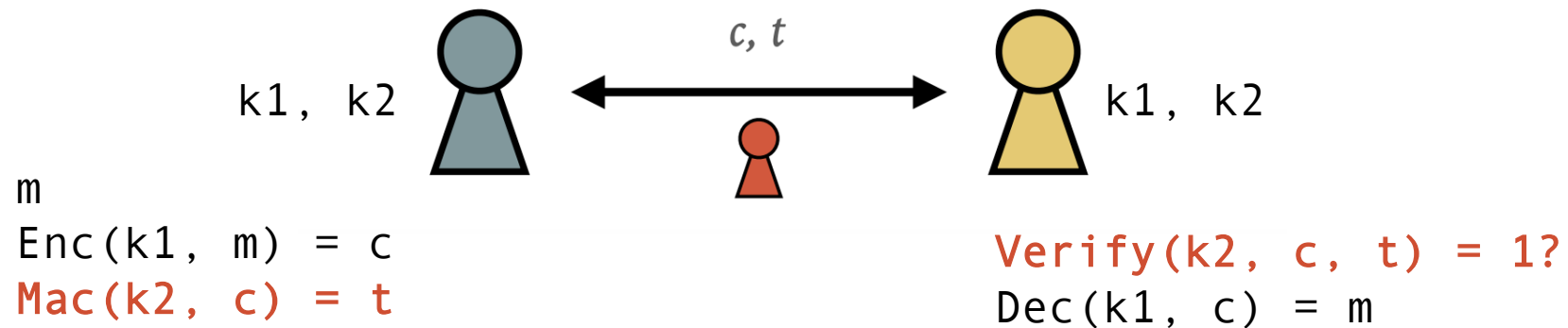


Encrypt then Authenticate: Is it secure?

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

Encrypt then authenticate

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

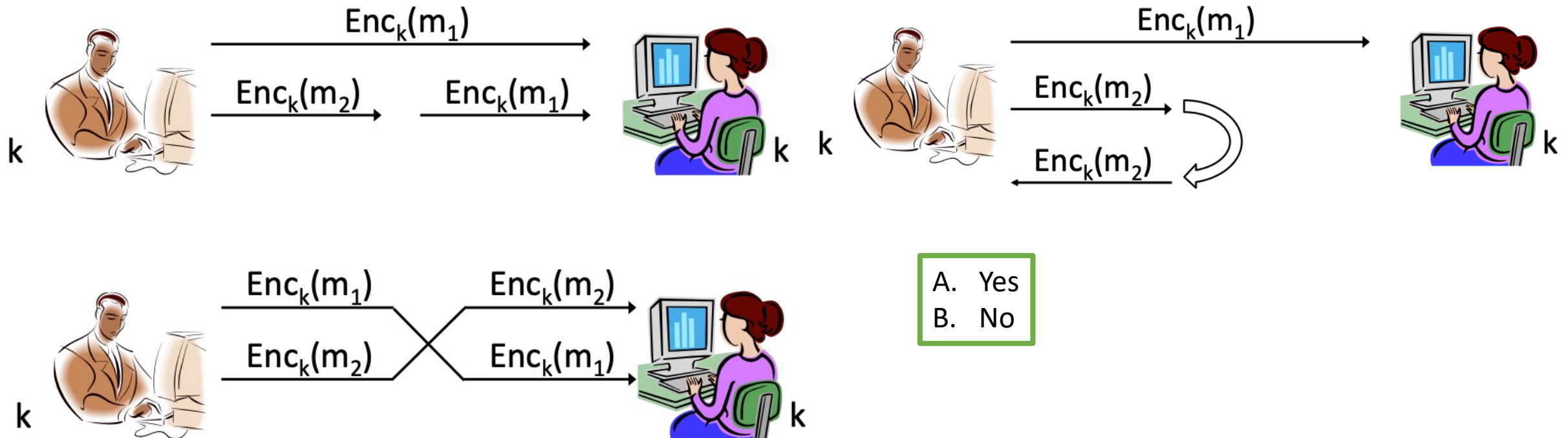


Encrypt then Authenticate is it secure?
Yes! If Enc is CPA secure and MAC is secure.

Bonus: This is actually now CCA secure!

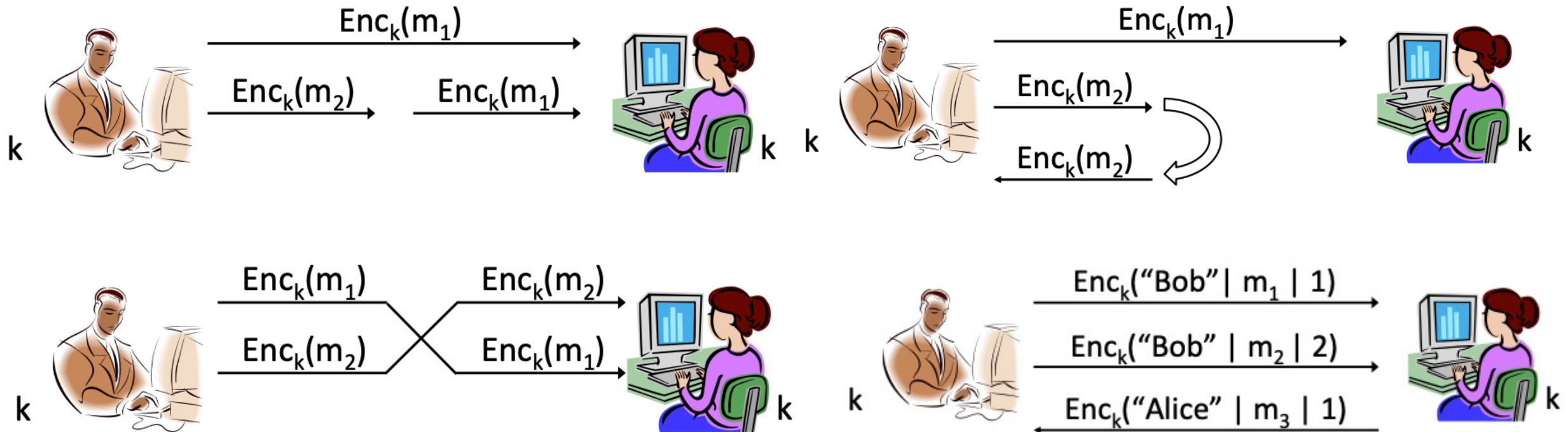
Secure Sessions: Consider parties who wish to communicate securely over the course of a session using authenticated encryption. Are they immune to the following attacks?

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

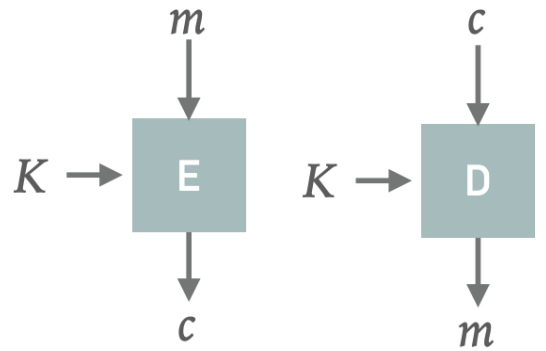


Secure Sessions: Consider parties who wish to communicate securely over the course of a session using authenticated encryption. Are they immune to the following attacks?

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



Symmetric Key Cryptography

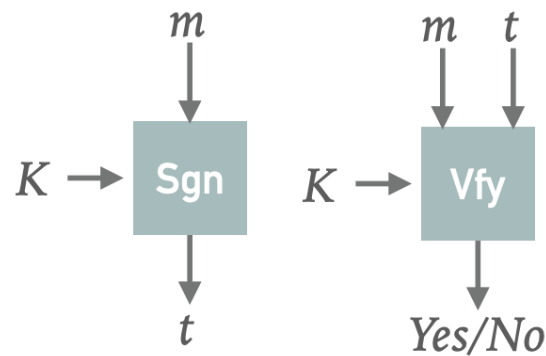


CONFIDENTIALITY

Block ciphers

Deterministic \Rightarrow use IVs

Fixed block size \Rightarrow use encryption "modes"

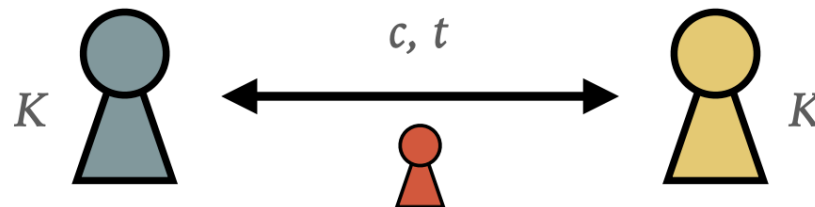


INTEGRITY

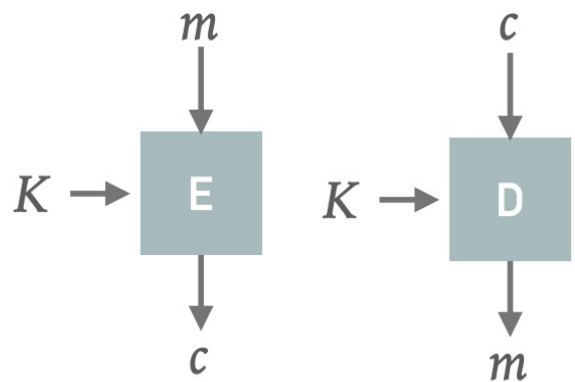
Message Authentication Codes (MACs)

Send (message, tag) pairs

Verify that they match



Symmetric Key Cryptography

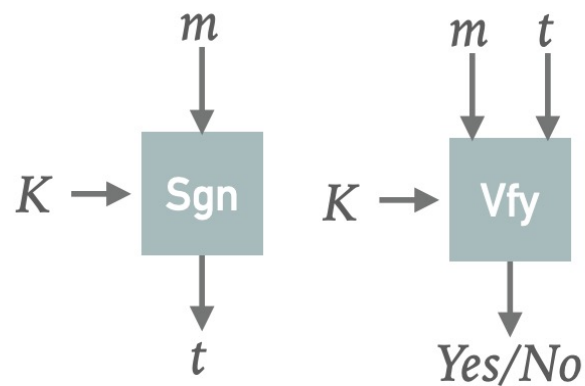


CONFIDENTIALITY

Block ciphers

Deterministic \Rightarrow use IVs

Fixed block size \Rightarrow use encryption "modes"



INTEGRITY

Message Authentication Codes (MACs)

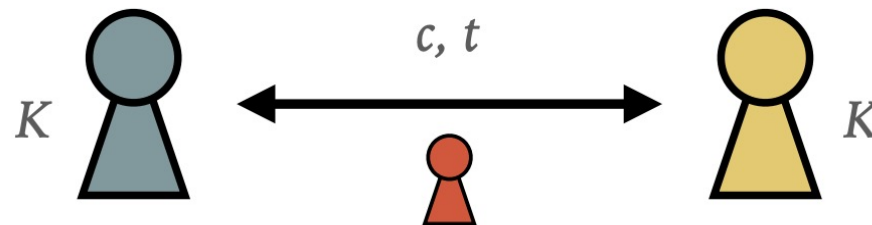
Send (message, tag) pairs

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

$$\{1, 2, \dots, N-1\} = \{g^0 \bmod N, g^1 \bmod N, \dots, g^{N-2} \bmod N\}$$

$$N=5, g=3$$

$$3^0 \bmod 5 = 1 \quad 3^1 \bmod 5 = 3 \quad 3^2 \bmod 5 = 4 \quad 3^3 \bmod 5 = 2$$


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 $g^x \bmod N$ it is *infeasible* to compute x
Discrete log problem

a, g, N
 $g^b \pmod N$



 g, N
 $g^a \pmod N$
 $g^b \pmod N$

g, N, b
 $g^a \pmod N$



Public knowledge: g and N

Pick random a

$g^a \pmod N$



$g^b \pmod N$




Pick random b

Compute $(g^b \pmod N)^a = g^{ab} \pmod N$

Compute $(g^a \pmod N)^b = g^{ab} \pmod N$



Shared secret: This is the key


$$g \ N$$
$$g^a \ \text{mod } N$$
$$g^b \ \text{mod } N$$

$$g^{ab} \ \text{mod } N$$

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

$$g^a \ \text{mod } N * g^b \ \text{mod } N = g^{a+b} \ \text{mod } N$$

Key property:

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

But what about *active intermediaries*?



g N

$g^a \bmod N$

$g^b \bmod N$

$g^{ab} \bmod N$

Given g and $g^x \bmod N$ it is *infeasible* to compute x
Discrete log problem

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

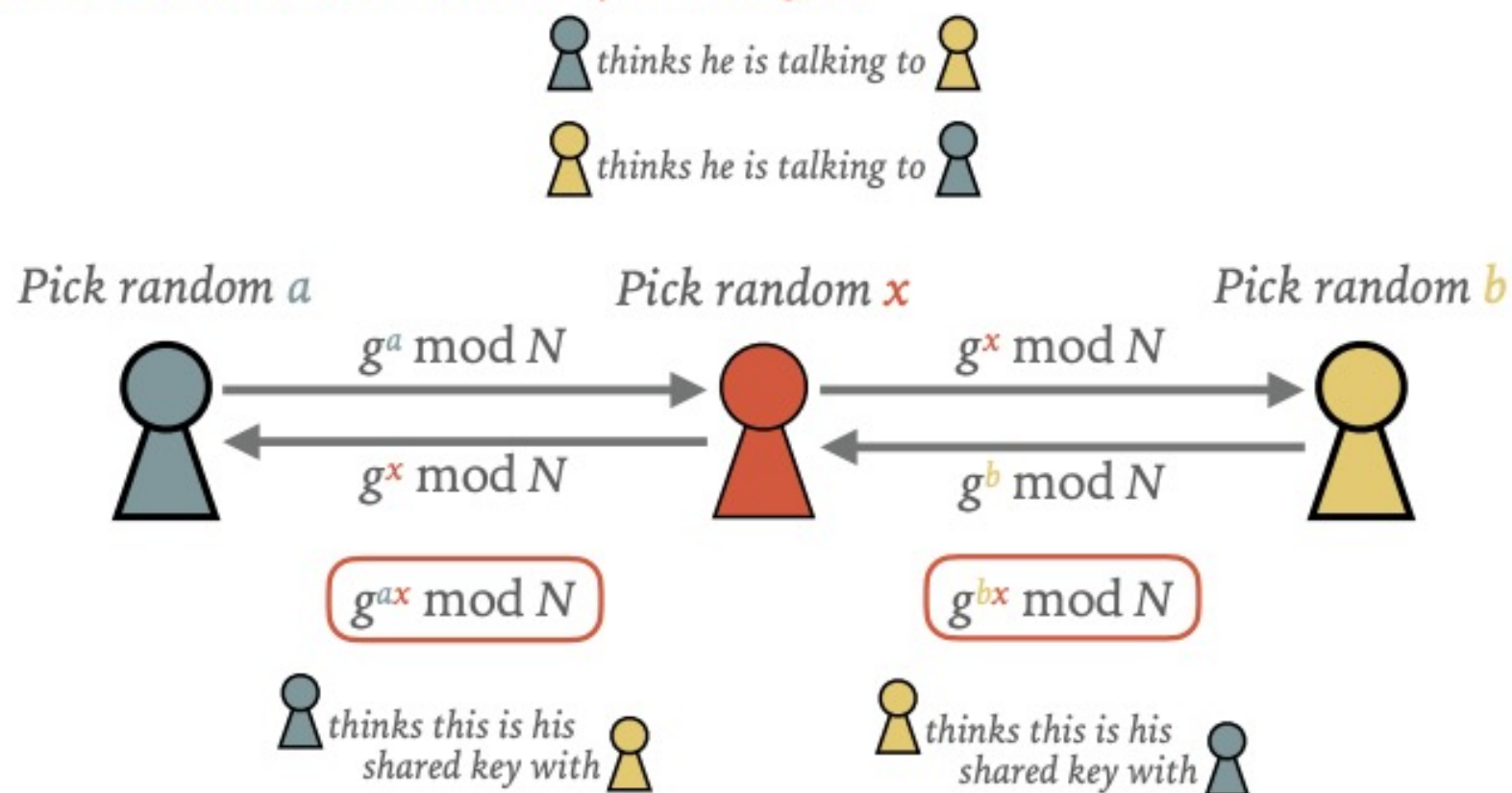
$$g^a \bmod N * g^b \bmod N = g^{a+b} \bmod N$$

Key property:

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

But what about *active intermediaries*?

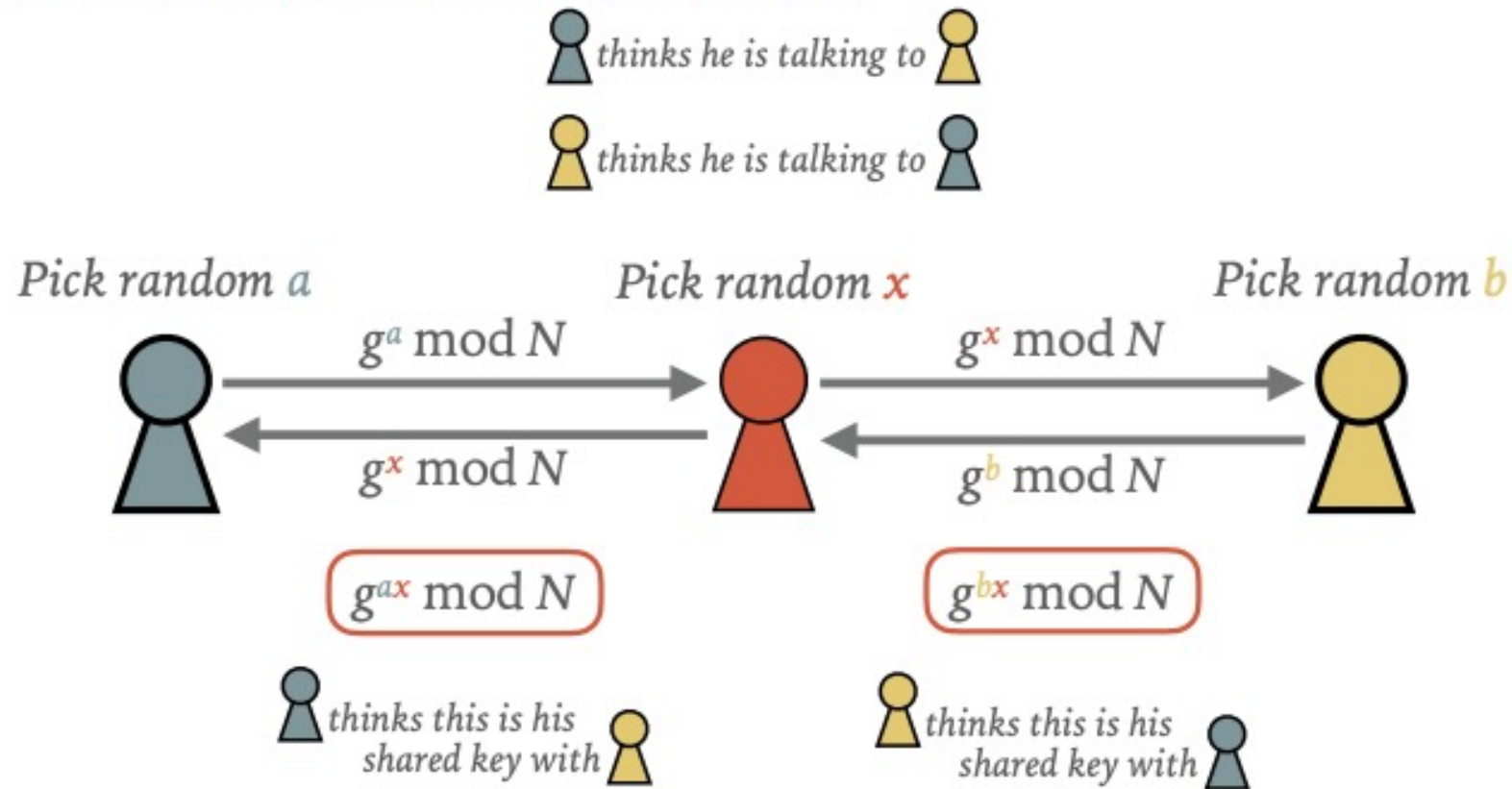
The attacker can interpose between the two communicating parties and insert, delete, and modify messages.



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

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



$pk = (N, e)$



$c = \text{Enc}_{pk}(m) = m^e \bmod N$



$d = \text{Dec}_{sk}(c) = c^d \bmod N$

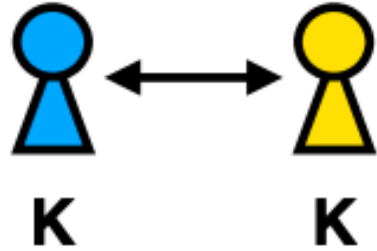
RSA Security

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

TO FIX THIS PROBLEM WE NEED...

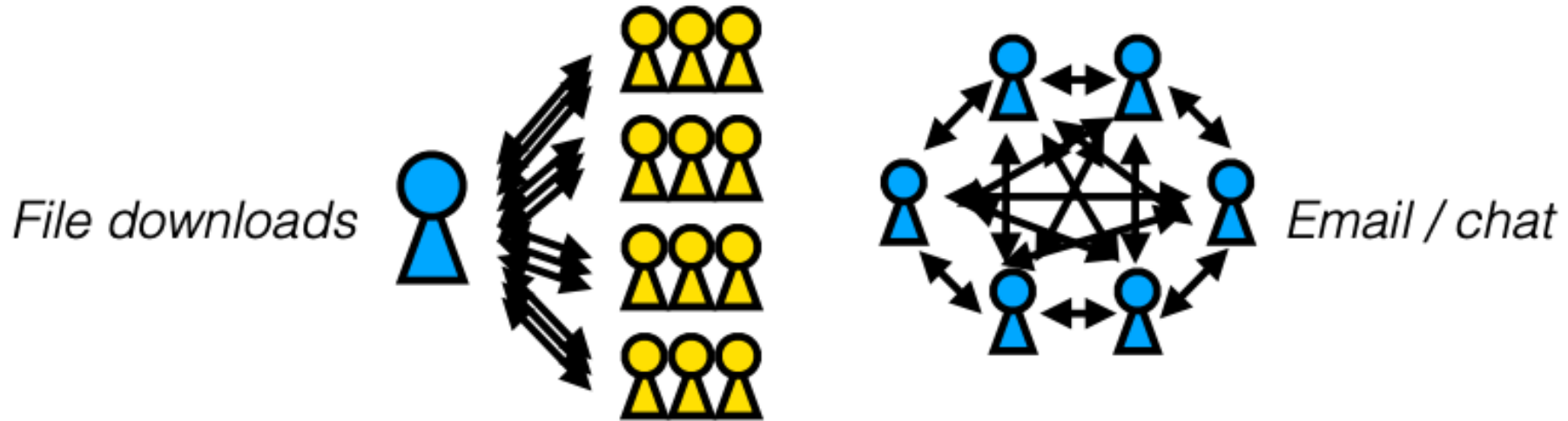
BLACKBOX #5:
PUBLIC KEY CRYPTOGRAPHY

Shortcomings of symmetric key



Establishing a pairwise key requires a **key exchange**, which requires both parties to be **online**

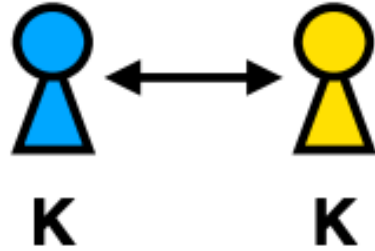
Issue #1: Requires pairwise key exchanges



One-to-many:
 $O(N)$ key exchanges

All-to-all:
 $O(N^2)$ key exchanges

Shortcomings of symmetric key



Establishing a pairwise key requires a **key exchange**, which requires both parties to be **online**

Issue #1: Requires *pairwise* key exchanges

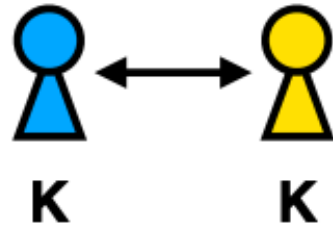


One-to-many:
 $O(N)$ key
exchanges

Blue user uploads a document, then goes offline (e.g., forever)

Later, a yellow user wants to get a copy; how can it know the copy is really from the blue user?

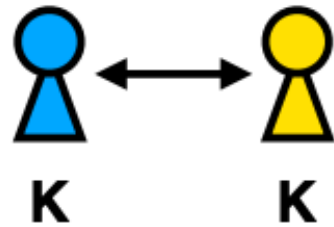
Shortcomings of symmetric key



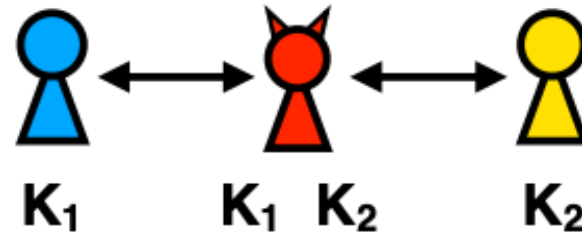
Establishing a pairwise key requires a **key exchange**, which requires both parties to be **online**

Issue #3: How do you know to whom you're talking?

Diffie-Hellman is resilient to *eavesdropping*, but **not tampering**



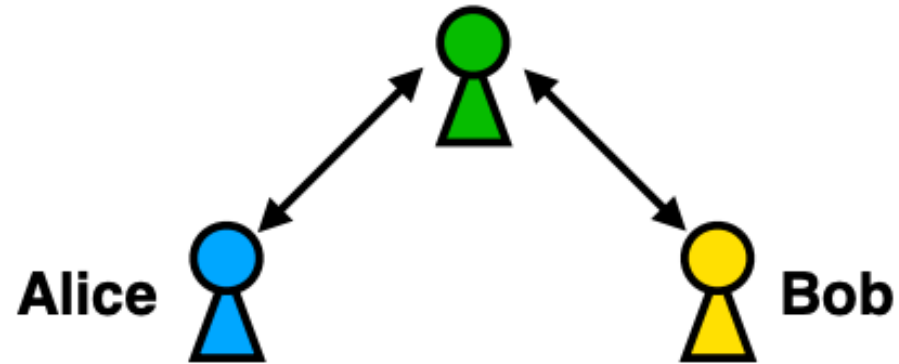
vs



Trusted Third Party

A protocol that solves this with ***trust***

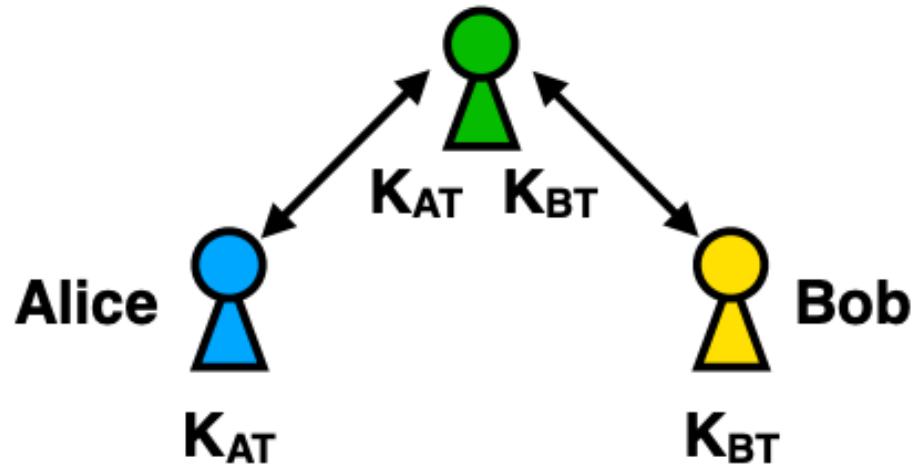
Trent: *A trusted third party*



Trusted Third Party

A protocol that solves this with ***trust***

Trent: *A trusted third party*

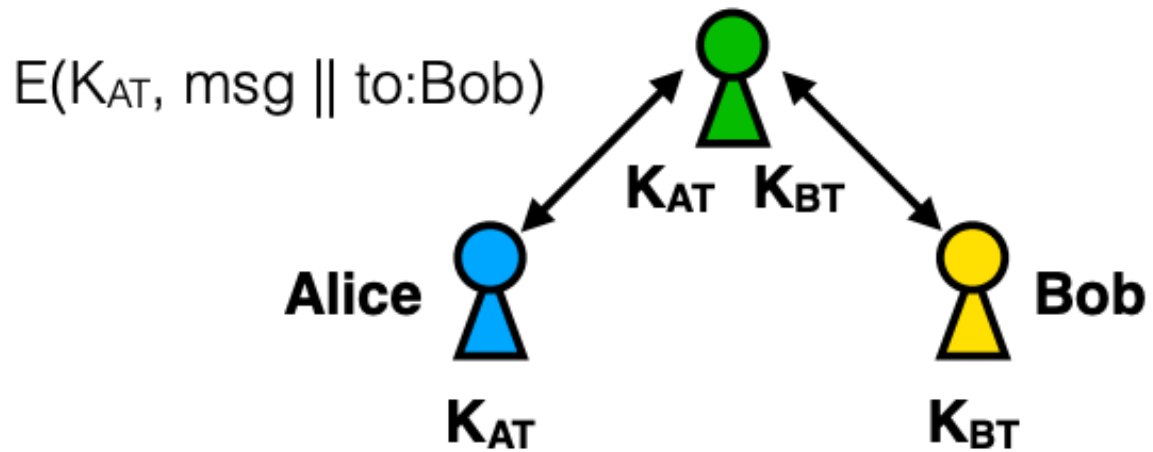


1. Everybody establishes a pairwise key with Trent
Good: $O(N)$ key exchanges

Trusted Third Party

A protocol that solves this with ***trust***

Trent: *A trusted third party*



1. Everybody establishes a pairwise key with Trent

Good: $O(N)$ key exchanges

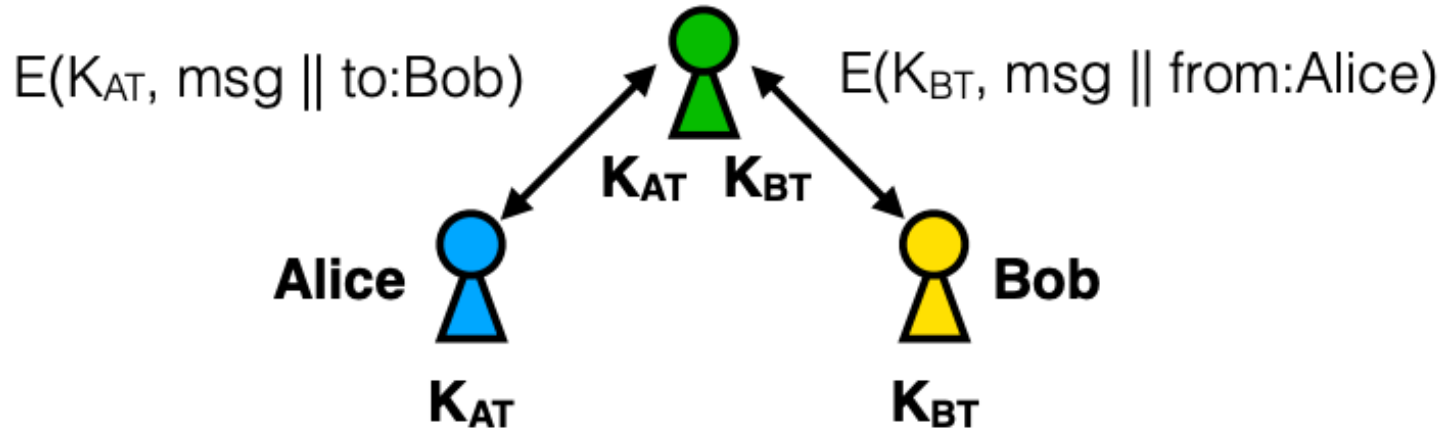
2. Trent validates each user's identity; includes in message

Good: *Authenticated communication*

Trusted Third Party

A protocol that solves this with **trust**

Trent: *A trusted third party*



1. Everybody establishes a pairwise key with Trent

Good: $O(N)$ key exchanges

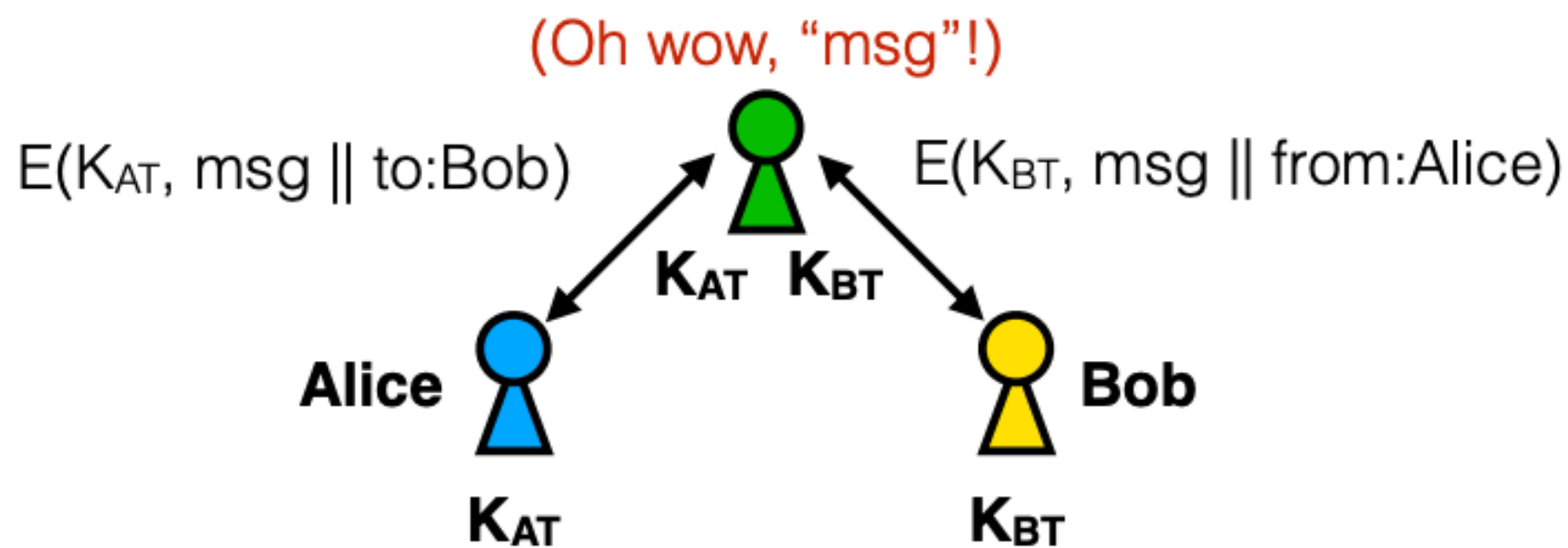
2. Trent validates each user's identity; includes in message

Good: *Authenticated communication*

Bad: All messages get sent through Trent

What are we trusting Trent not to do?

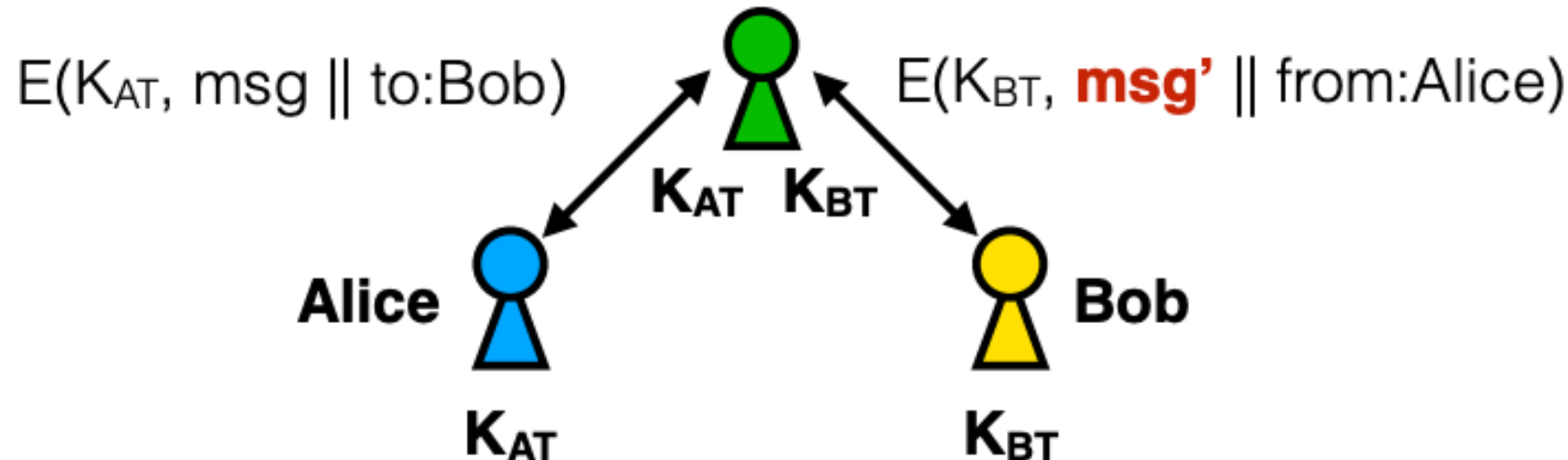
Just as “secure” meant nothing without an attack model,
“trusted” means nothing without a **trust model**



1. Do not *read* messages

What are we trusting Trent not to do?

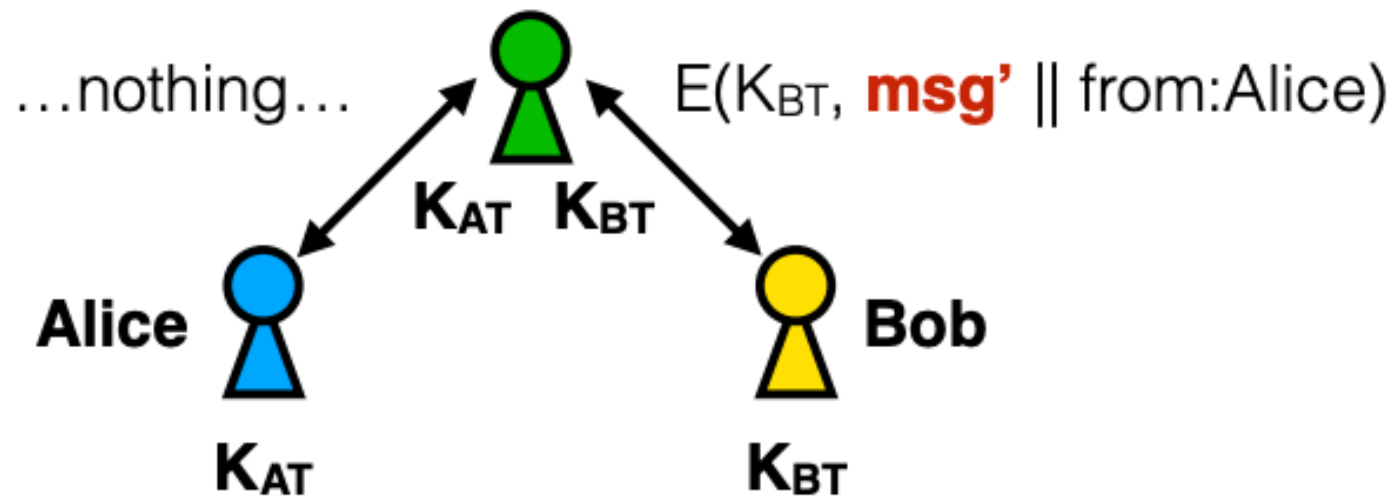
Just as “secure” meant nothing without an attack model,
“trusted” means nothing without a **trust model**



1. Do not *read* messages
2. Do not *alter* messages

What are we trusting Trent not to do?

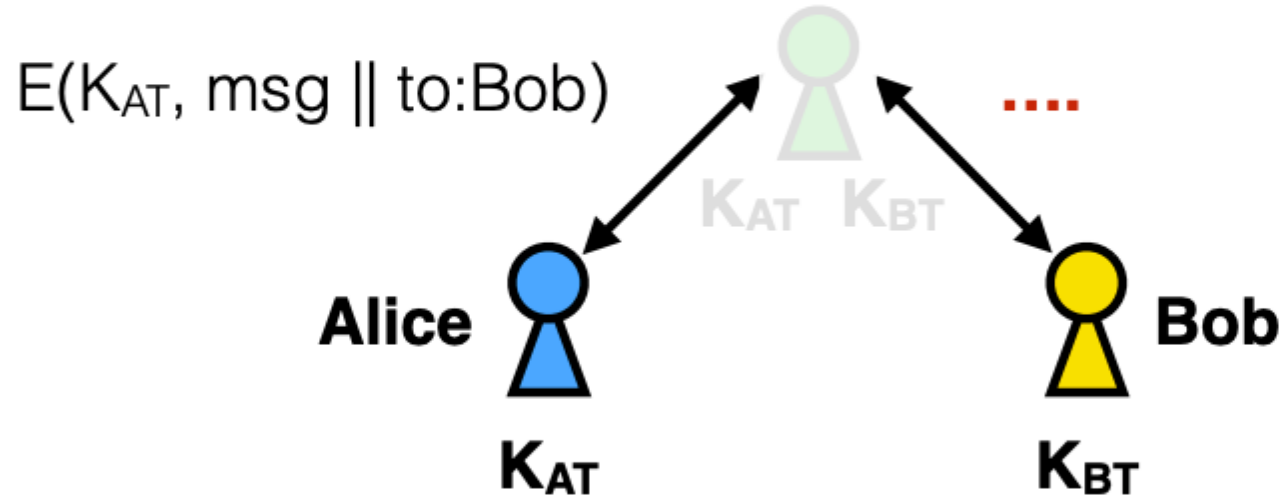
Just as “secure” meant nothing without an attack model,
“trusted” means nothing without a **trust model**



1. Do not *read* messages
2. Do not *alter* messages
3. Do not *forge* messages

What are we trusting Trent not to do?

Just as “secure” meant nothing without an attack model,
“trusted” means nothing without a **trust model**



1. Do not *read* messages
2. Do not *alter* messages
3. Do not *forge* messages
4. Do not *go offline*

Public key encryption

A public key encryption scheme comprises three algorithms

Key generation G

→ $PK =$ **public key**

→ $SK =$ **secret key**

Encryption $E(PK, m)$

→ cipher text c

Decryption $D(SK, c)$

→ original msg

Correctness

$$D(SK, E(PK, m)) = m$$

Security

$E(PK, m)$ should appear random
(small change to (PK, m) leads
to large changes to c)

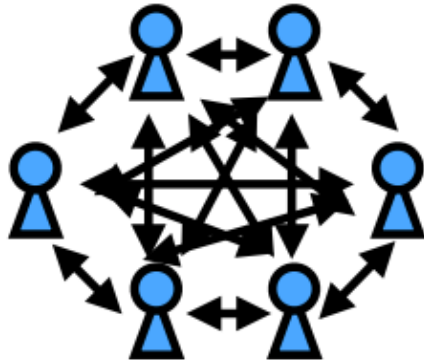
$E()$ should approximate a one-way
trapdoor function: cannot invert
without access to SK

Protocols with public key encryption

Goal: deliver a confidential message

Symmetric key

Email / chat



All-to-all:
 $O(N^2)$ key
exchanges



Generate public/private
key pair (PK,SK)

Announce PK publicly
(on website, in newspaper, ...)

Obtain PK



Send $c = E(\text{PK}, \text{msg})$



Decrypt $D(\text{SK}, c) = \text{msg}$

$O(N)$ keys in total

Overcoming fixed message sizes

Encryption $E(\text{PK}, \text{msg})$


- Inputs
 - **Public** key PK
 - Message msg of *fixed size*
- Outputs: a cipher text c
same size as msg

Like block ciphers,
but there are not
“modes” of public
key encryption

Public key operations are *sloooooow!*

Symmetric key operations are fast

Hybrid encryption

 Generate public/private key pair (PK,SK); publicize PK

Obtain PK



Generate *symmetric* key K

Symm key Compute $c_{\text{msg}} = e(K, \text{msg})$

Public key Compute $c_K = E(\text{PK}, K)$

Send $c_K \parallel c_{\text{msg}}$



Decrypt $D(\text{SK}, c_K) = K$ *Public key*

Decrypt $d(K, c_{\text{msg}}) = \text{msg}$ *Symm key*

Hybrid encryption

Obtain PK

Generate *symmetric* key K

Compute $c_{\text{msg}} = e(K, \text{msg})$

Compute $c_K = E(\text{PK}, K)$

Send $c_K \parallel c_{\text{msg}}$



The easy key distribution of public key

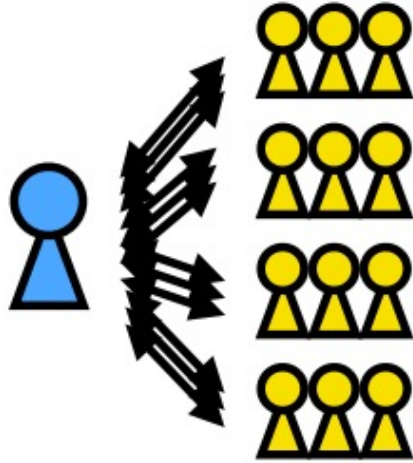
The speed and arbitrary message length of symmetric key

Protocols with public key cryptography

Goal: determine from whom a message came

Symmetric key

File downloads



One-to-many:
 $O(N)$ key
exchanges

Digital signatures

A digital signature scheme comprises two algorithms

Signing function $Sgn(SK, m)$

- Inputs
 - **Secret** key SK
 - Fixed-length message
- Outputs: a *signature* s

This is a *randomized* algorithm
(nondeterministic output)

SK a.k.a. “Signing key”

**Only one person can sign with
a given (PK,SK) pair**

Verification function $Vfy(PK, m, s)$

- Inputs
 - **Public** key PK
 - Message and signature
- Outputs: Yes/No if valid (m,s)

Deterministic algorithm

**Anyone with the PK
can verify**

Digital signatures

A digital signature scheme comprises two algorithms

Signing **Sgn(SK, m)**

→ a signature *s*

Verification **Vfy(PK, m, s)**

→ Yes/No if valid (m,s)

Correctness

$Vfy(PK, m, Sgn(SK, m)) = \text{Yes}$

Security

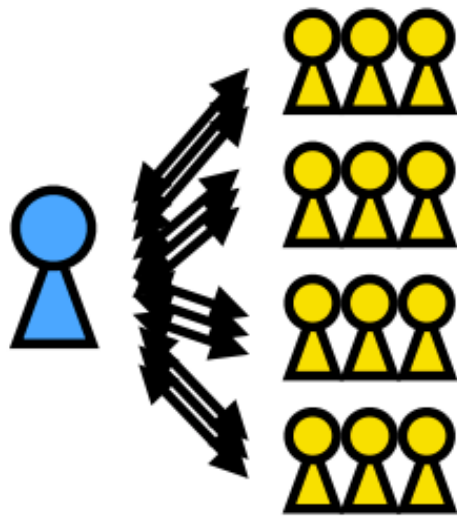
Same as with MACs: even after a chosen plaintext attack, the attacker cannot demonstrate an existential forgery

Protocols with digital signatures

Goal: determine from whom a message came

Symmetric key

File downloads



One-to-many:
 $O(N)$ key
exchanges



Generate public/private
key pair (PK,SK)

Announce PK publicly
(on website, in newspaper, ...)

Compute $\text{sig} = \text{Sgn}(\text{SK}, \text{msg})$

Publish $\text{msg} \parallel \text{sig}$

can now go offline!

Digital signature properties

Authenticity

Bob can prove that a message signed by Alice is truly from Alice (even without a *pairwise* key)

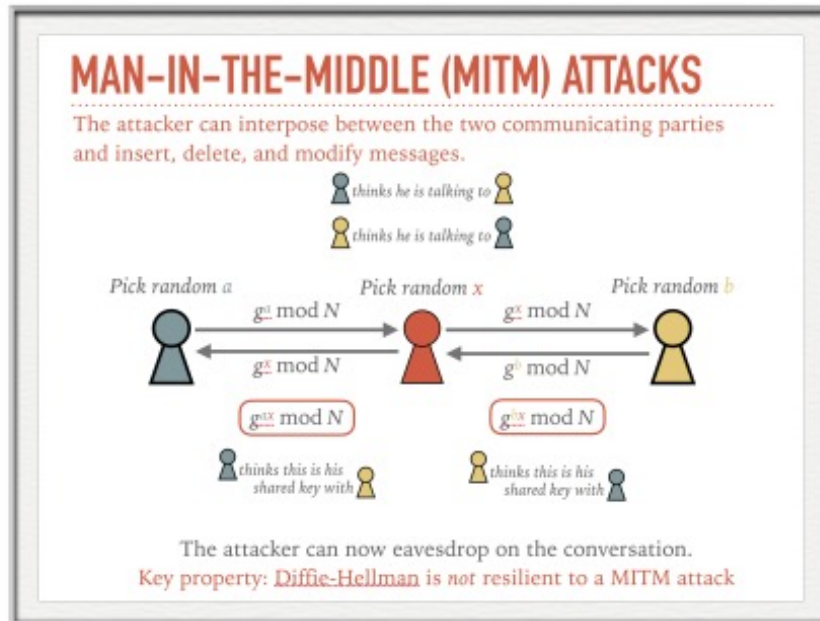
Integrity

Bob can prove that no one has tampered with a signed message

Non-repudiation

Once Alice signs a message, she cannot subsequently claim she did *not* sign that message

RECALL OUR PROBLEM WITH DIFFIE-HELLMAN




The two communicating parties thought, *but did not confirm*, that they were talking to one another.


Therefore, they were vulnerable to MITM attacks.

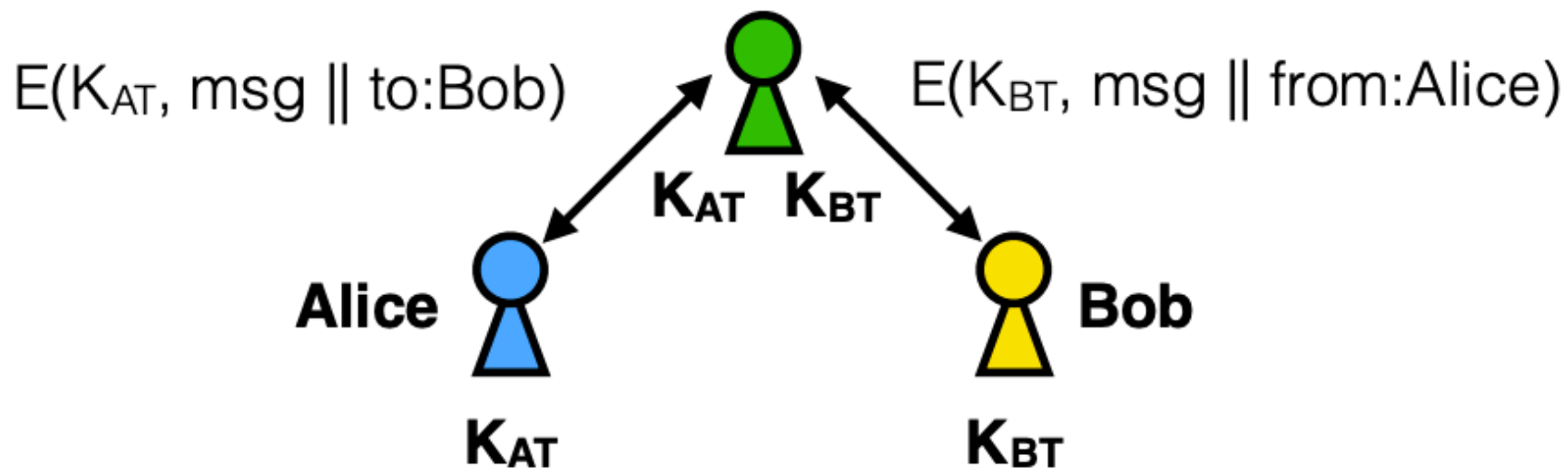
Certificates allow us to verify with whom we are communicating.

We will solve this by incorporating public key cryptography

Back to authentication

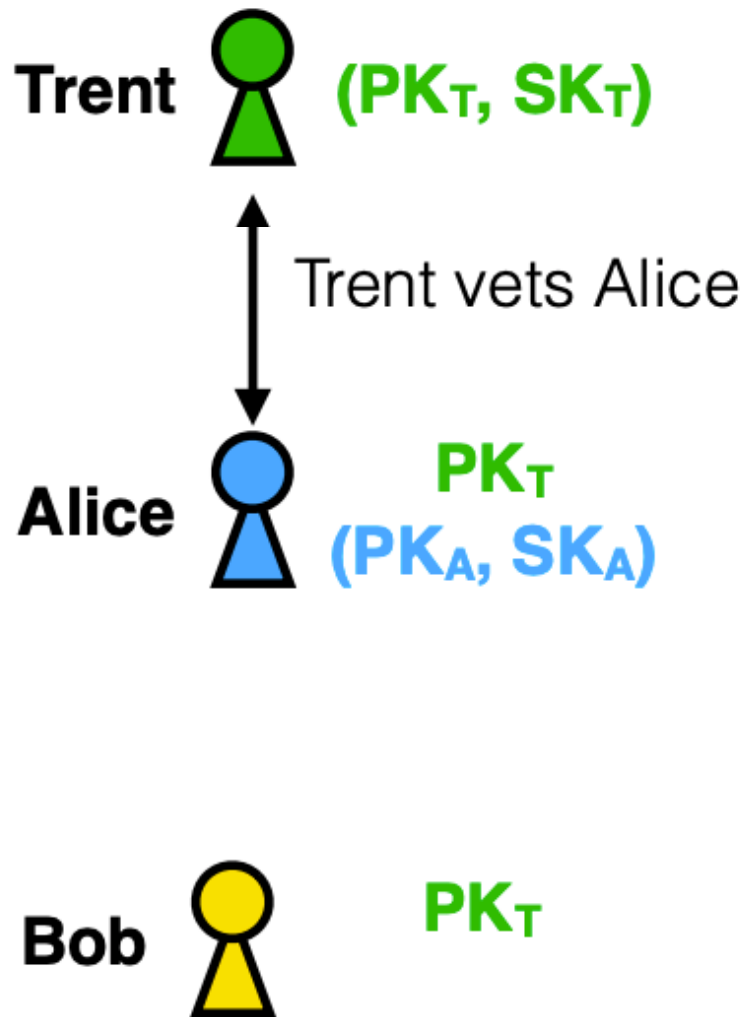
 Generate public/private key pair (PK,SK); publicize PK

How can we know it was really  who posted PK?



Can we achieve authentication without Trent in the middle of *every message*?

Authentication with public keys



1. Trent's public key is widely disseminated (pre-installed in browsers/operating systems)

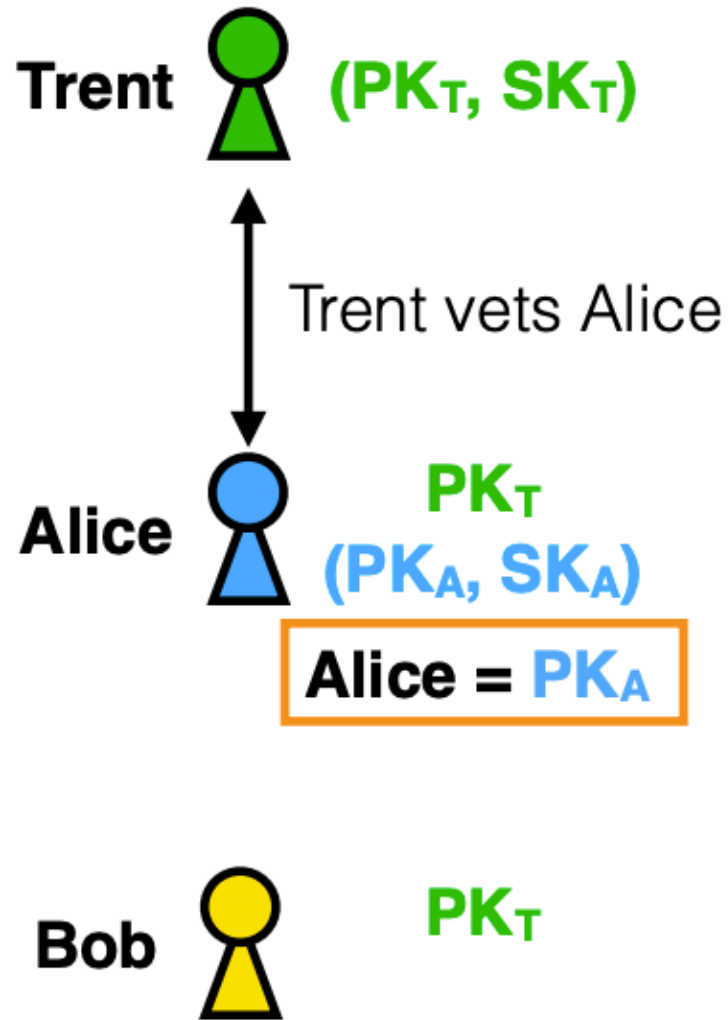
2. Alice generates a public/private key pair and asks Trent to bind her PK_A to her identity

3. Trent *signs* a message (with SK_T):

"The owner of the secret key corresponding to PK_A is Alice"

This message + sig = **Certificate**

Authentication with public keys



1. Trent's public key is widely disseminated (pre-installed in browsers/operating systems)

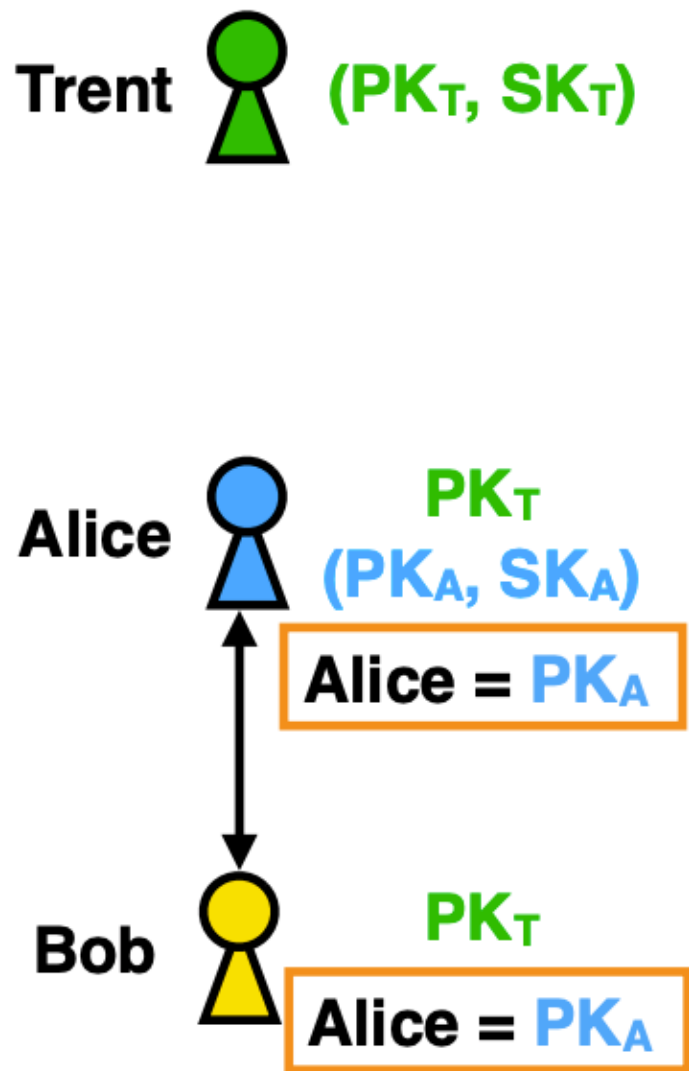
2. Alice generates a public/private key pair and asks Trent to bind her PK_A to her identity

3. Trent *signs* a message (with SK_T):

"The owner of the secret key corresponding to PK_A is Alice"

This message + sig = **Certificate**

Authentication with public keys



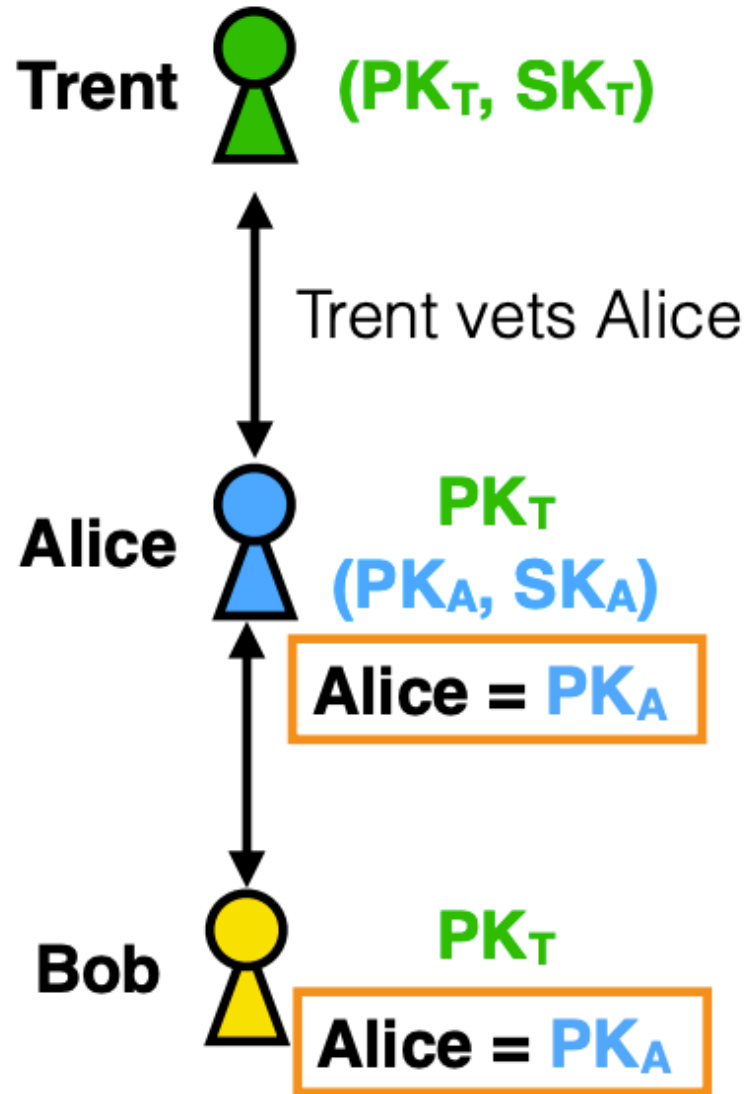
4. Alice makes her **certificate** publicly available (or Bob simply asks for it)

5. Bob verifies the **certificate** using PK_T

If Bob trusts Trent, then Bob trusts that he properly vetted Alice, and thus that her public key is PK_A

6. Bob (via hybrid encryption) sends a message to Alice using her public key PK_A

Authentication with public keys

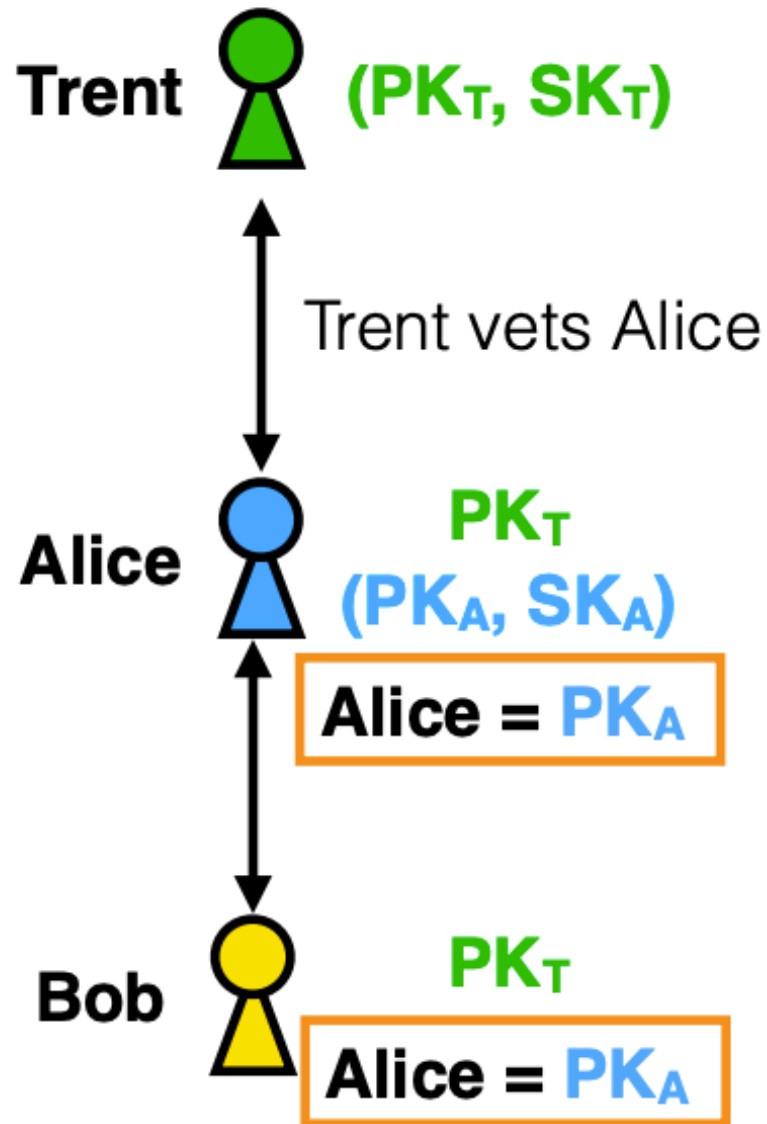


Properties

Trent need be online only when giving out **certificates**, not any time users want to communicate with one another

Alice and Bob can communicate in an authenticated manner without having to go through Trent

Authentication with public keys



Trust assumptions from our symmetric key protocol:

1. Do not *read* messages
2. Do not *alter* messages
3. Do not *forge* messages
4. Do not *go offline*

Trust assumptions in this public key protocol:

1. Correctly vet users
(Some more in practice...)

Certificate revocation

3. Trent *signs* a message (with SK_T):

“The owner of the secret key
corresponding to PK_A is Alice”

This message + sig = **Certificate**

Put another way:

“The only person who knows SK_A is Alice”

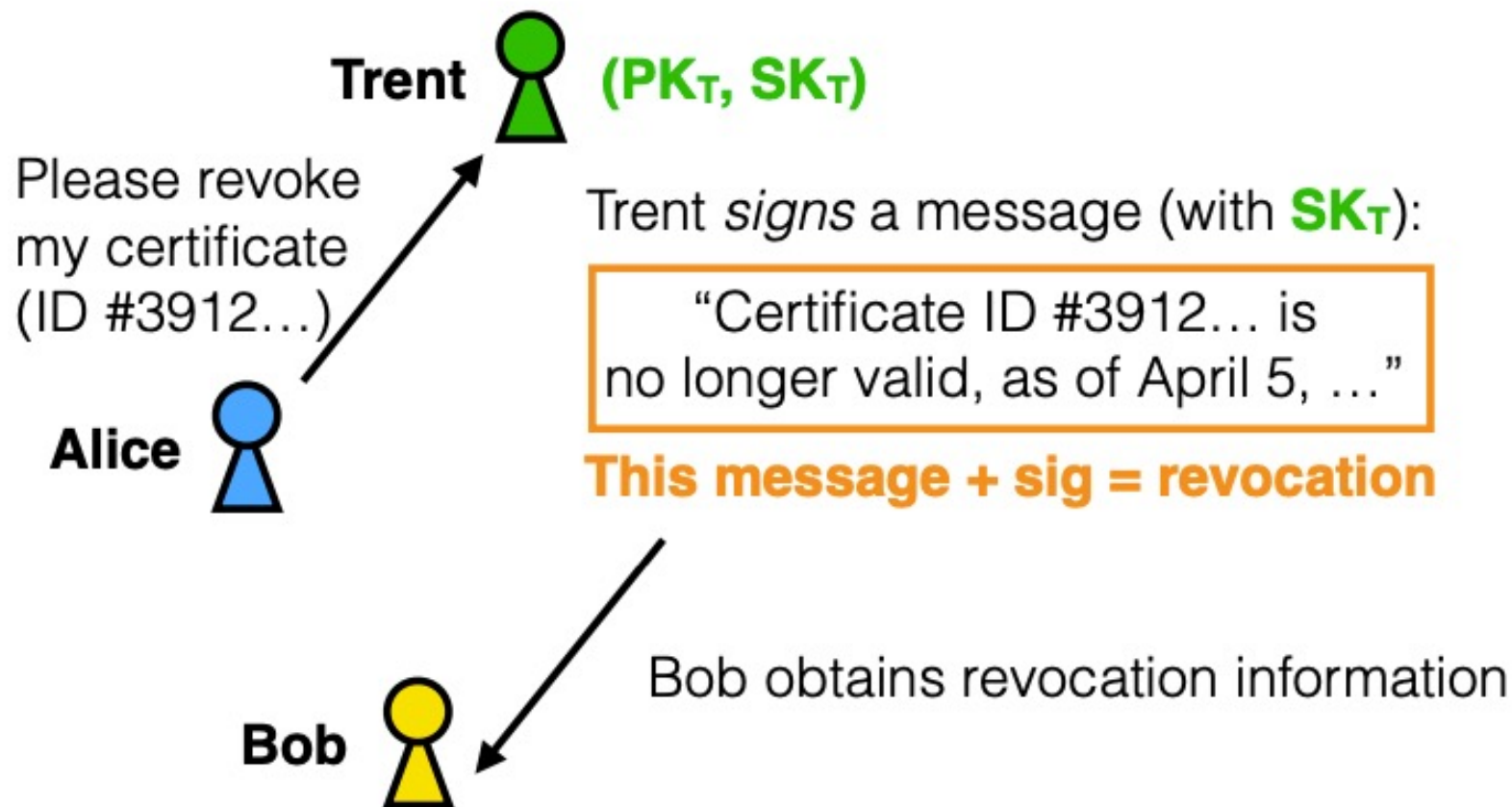
What happens if Alice's key gets compromised?

(Stolen, accidentally revealed, ...)

Certificate revocation



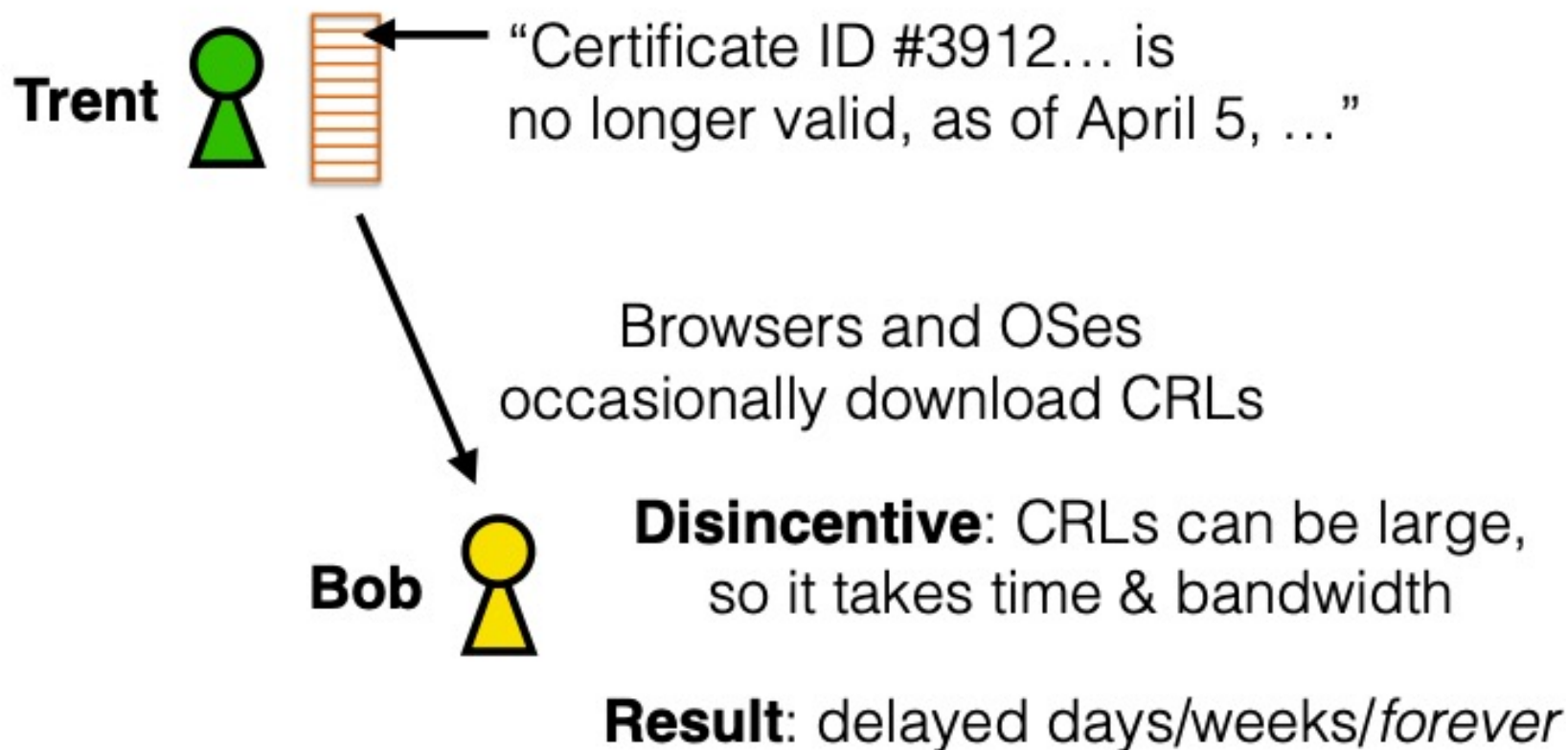
Certificate revocation



Obtaining revocation data

Certificate Revocation Lists (CRLs)

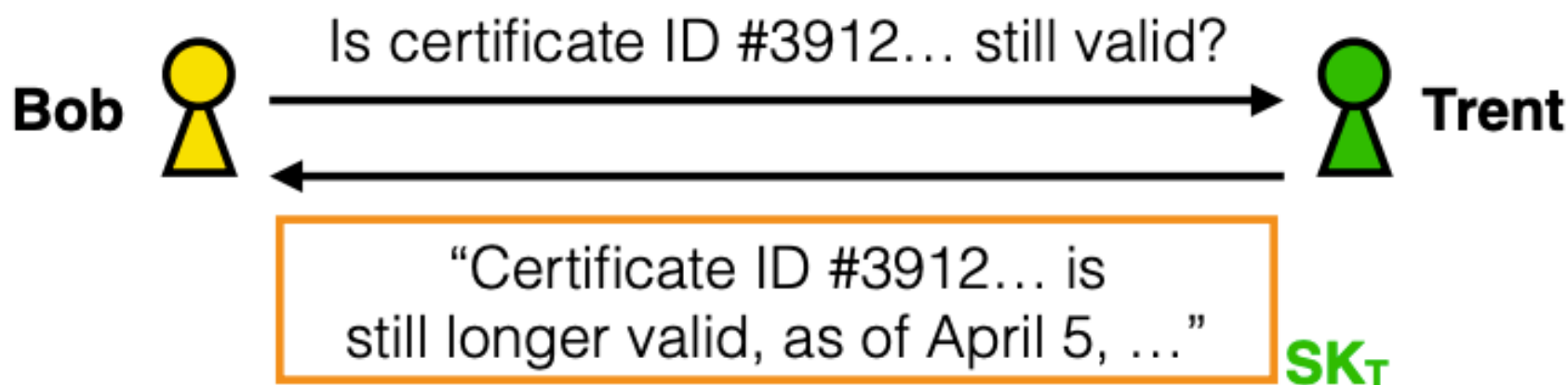
A (often large) signed list of revocations



Obtaining revocation data

Online Certificate Status Protocol (OCSP)

Browsers and OSes perform OCSP checks on-demand (when verifying the certificate)

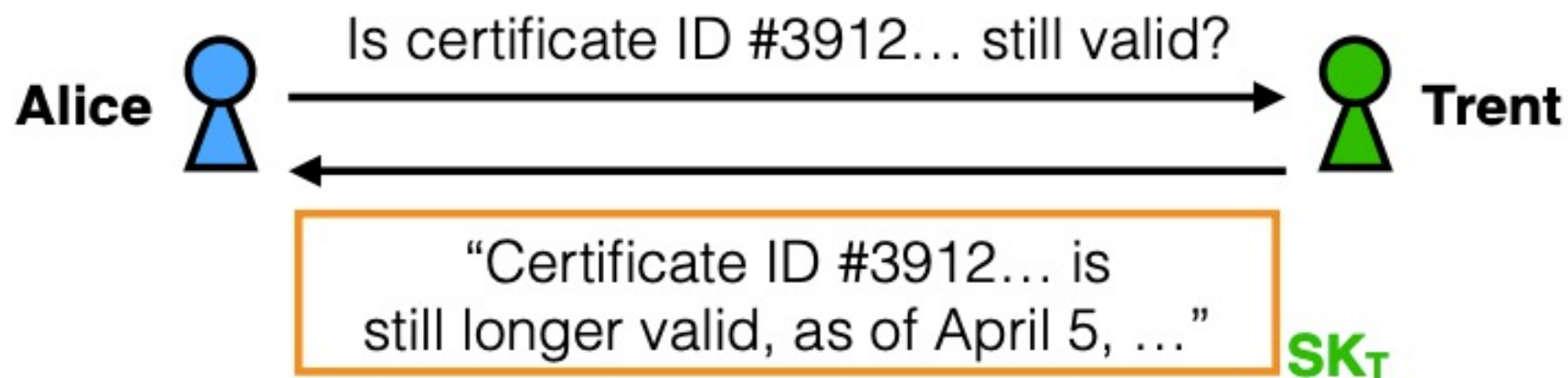


Disincentive: Still delays the initial validation of the certificate (can increase webpage load time)

Obtaining revocation data

OCSP Stapling

Websites issue OCSP requests,
include responses in initial handshake



**Alice forwards this to Bob along with
the certificate when they first
start to communicate**

Certificate revocation responsibilities



Alice's responsibility:

Request revocations



Trent's responsibility:

Make revocations publicly available

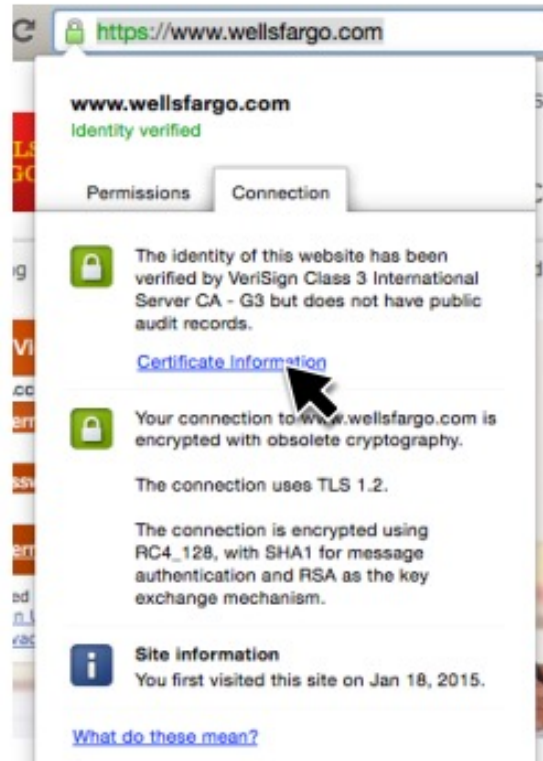


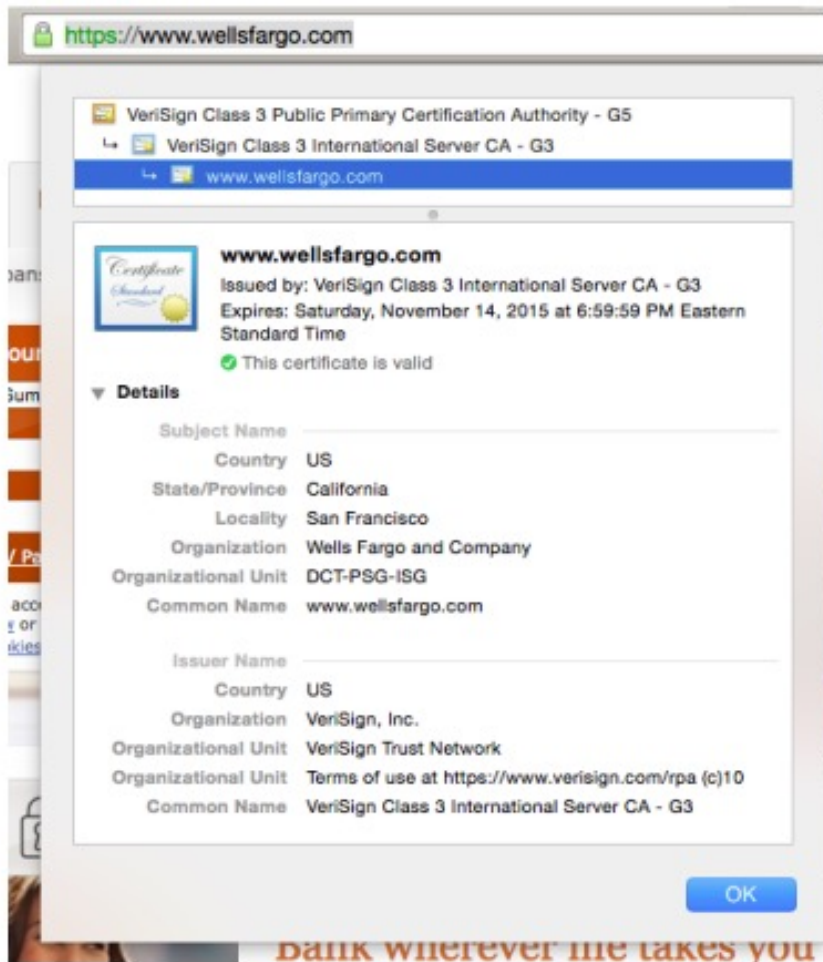
Bob's responsibility:

Check for revocations

Certificates in the wild

The lock icon indicates that the browser was able to authenticate the other end, i.e., validate its certificate





Certificate chain

Subject (who owns the public key)

Common name: the URL of the subject

Issuer (who verified the identity and signed this certificate)