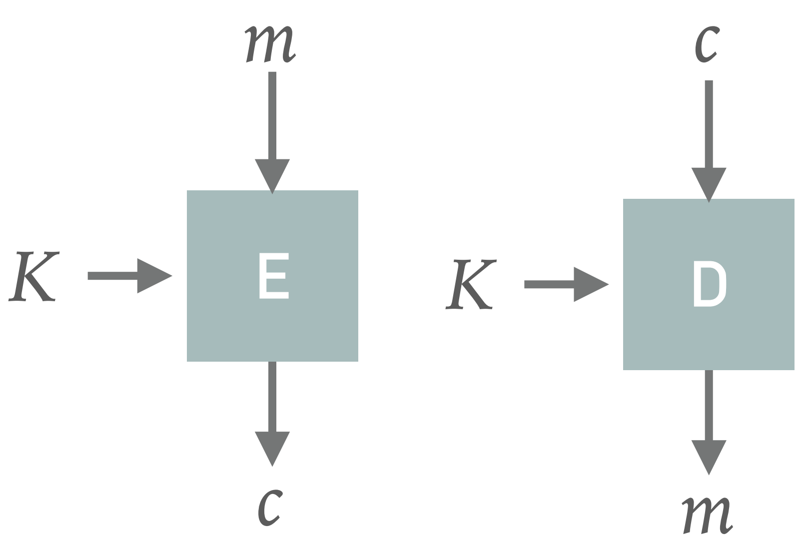
PUBLIC KEY CRYPTO

CMSC 414

MAR 27 2018

RECAP: SYMMETRIC KEY CRYPTO

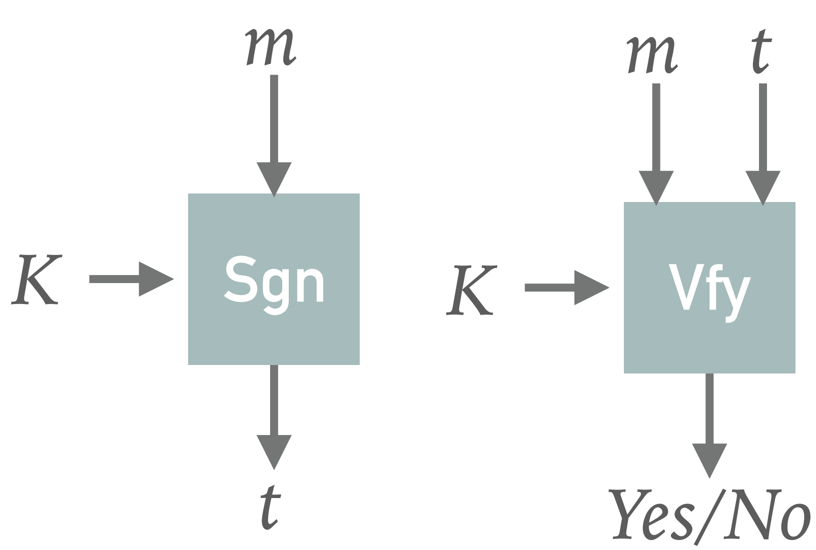


CONFIDENTIALITY

Block ciphers

Deterministic ⟹ use IVs

Fixed block size ⟹ use encryption "modes"

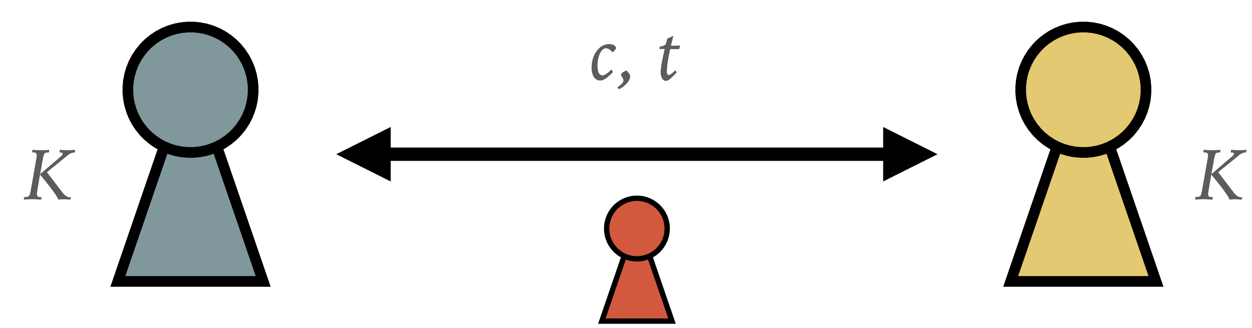


INTEGRITY

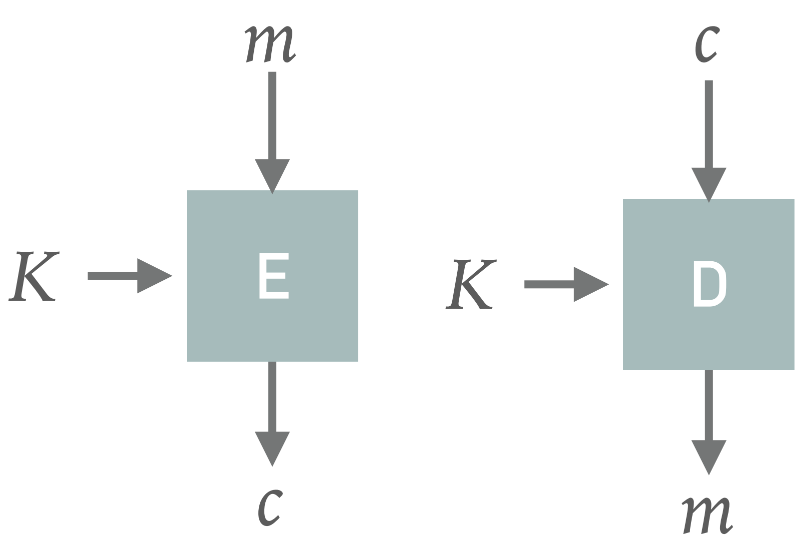
Message Authentication Codes (MACs)

Send (message, tag) pairs

Verify that they match



RECAP: SYMMETRIC KEY CRYPTO

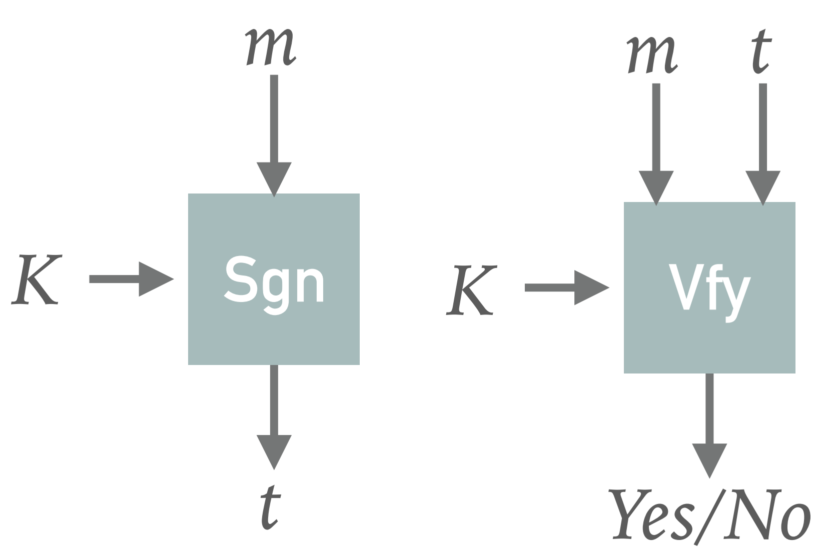


CONFIDENTIALITY

Block ciphers

Deterministic $\Longrightarrow$ use IVs

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



INTEGRITY

Message Authentication Codes (MACs)

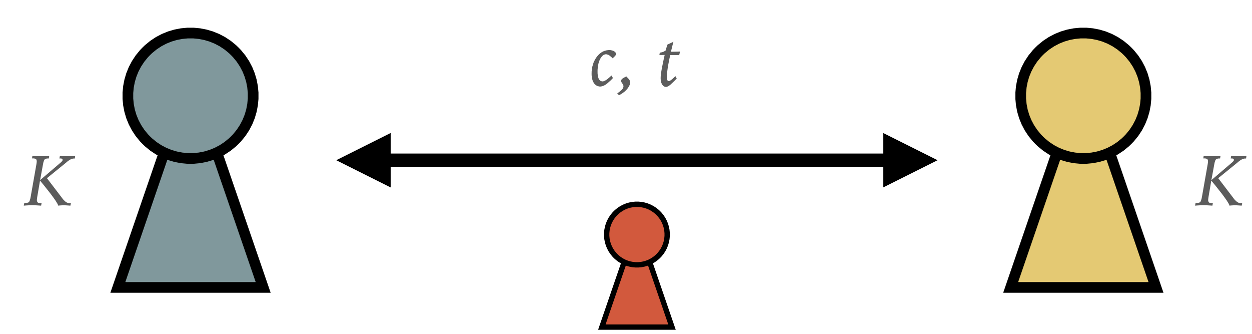
Send (message, tag) pairs

Verify that they match

Today:

How do we establish K?

How do we know with whom we are communicating?



BLACKBOX #4:
DIFFIE HELLMAN KEY ESTABLISHMENT

HIGH-LEVEL REVIEW OF MODULAR ARITHMETIC

$$x \bmod N$$

HIGH-LEVEL REVIEW OF MODULAR ARITHMETIC

$x \bmod N$

g is a generator of mod N if

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

HIGH-LEVEL REVIEW OF MODULAR ARITHMETIC

$x \bmod N$

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

$N{=}5, g{=}3$

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

HIGH-LEVEL REVIEW OF MODULAR ARITHMETIC

$x \bmod N$

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

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

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

HIGH-LEVEL REVIEW OF MODULAR ARITHMETIC

$x \bmod N$

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 $\log_g g^x$)

HIGH-LEVEL REVIEW OF MODULAR ARITHMETIC

$x \bmod N$

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

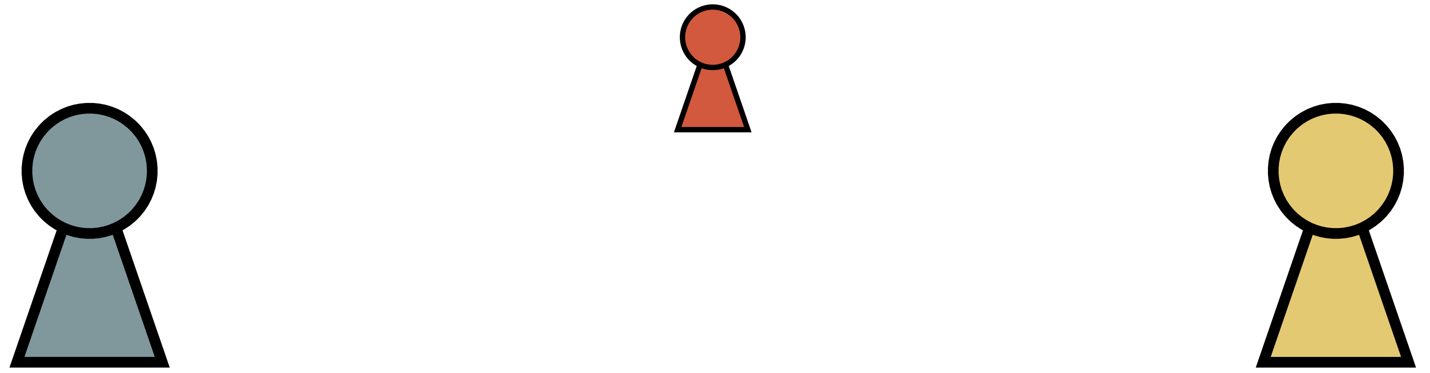
$N{=}5$, $g{=}3$
$3^0 \bmod 5 = 1$ $3^1 \bmod 5 = 3$ $3^2 \bmod 5 = 4$ $3^3 \bmod 5 = 2$

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

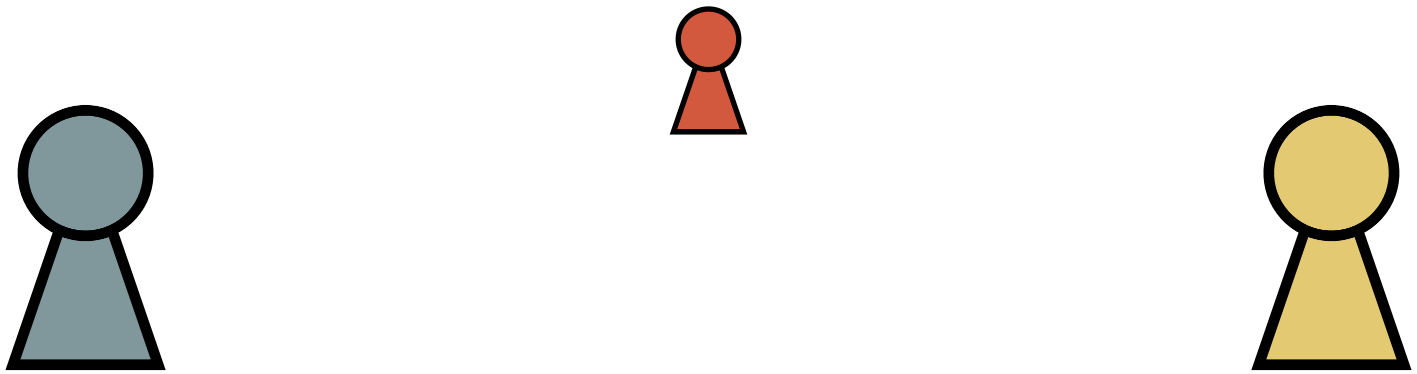
Given g and g^x, it is efficient to compute x
(simply take $\log_g g^x$)

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

DIFFIE-HELLMAN KEY EXCHANGE

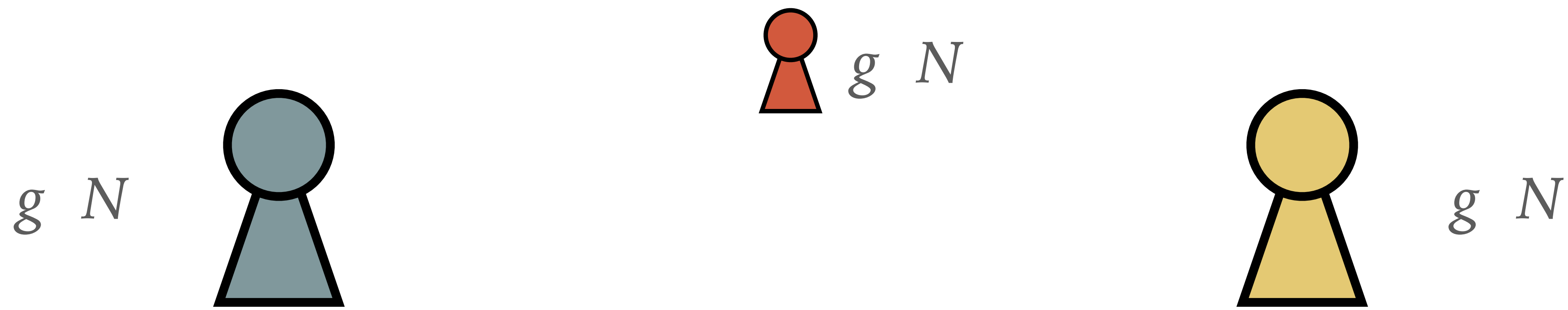


DIFFIE-HELLMAN KEY EXCHANGE



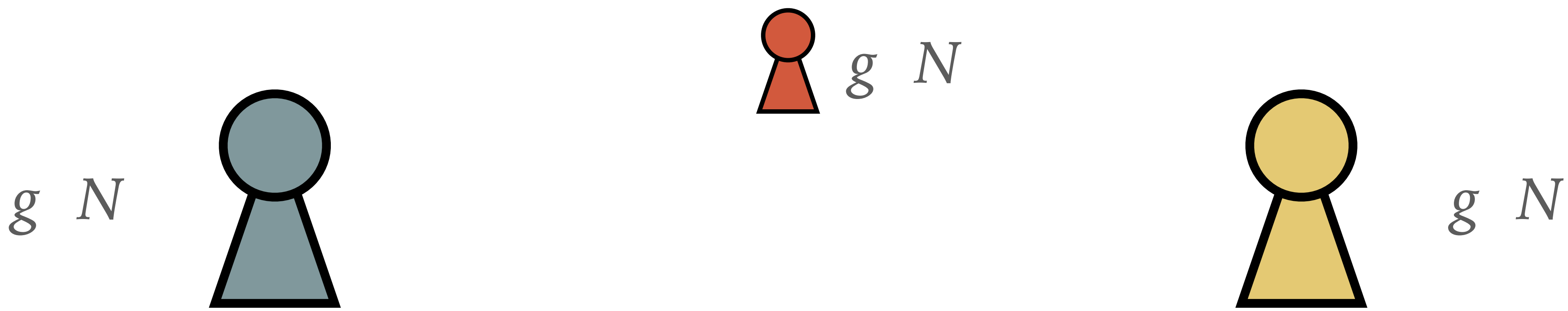
Public knowledge: g and N

DIFFIE–HELLMAN KEY EXCHANGE



Public knowledge: g and N

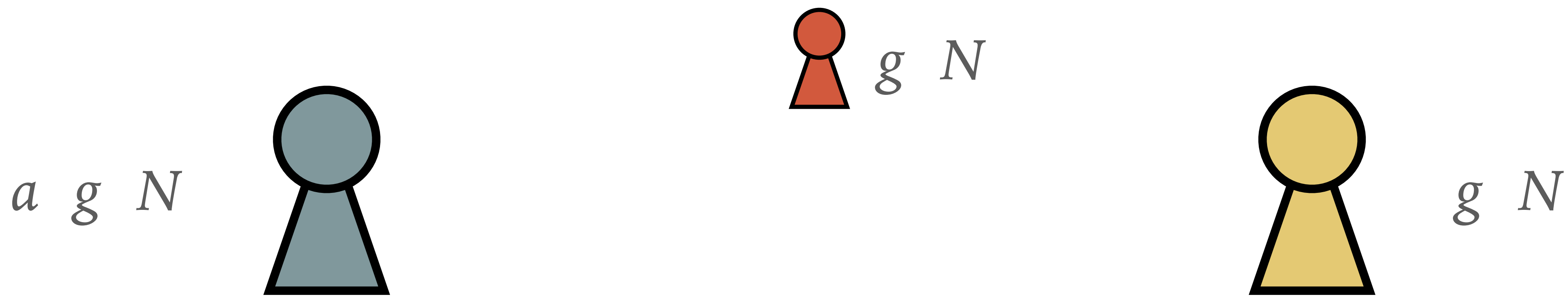
DIFFIE-HELLMAN KEY EXCHANGE



Public knowledge: g and N

Pick random a

DIFFIE-HELLMAN KEY EXCHANGE



Public knowledge: g and N

Pick random a

DIFFIE-HELLMAN KEY EXCHANGE

g N

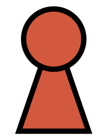
a g N | *g N*

Public knowledge: g and N

Pick random a

$$\xrightarrow{g^a \bmod N}$$

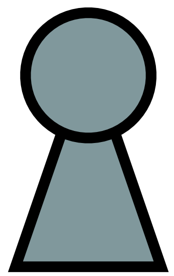
DIFFIE-HELLMAN KEY EXCHANGE

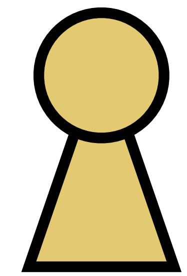


$g \quad N$

$g^a \bmod N$

$a \quad g \quad N$





$g \quad N$

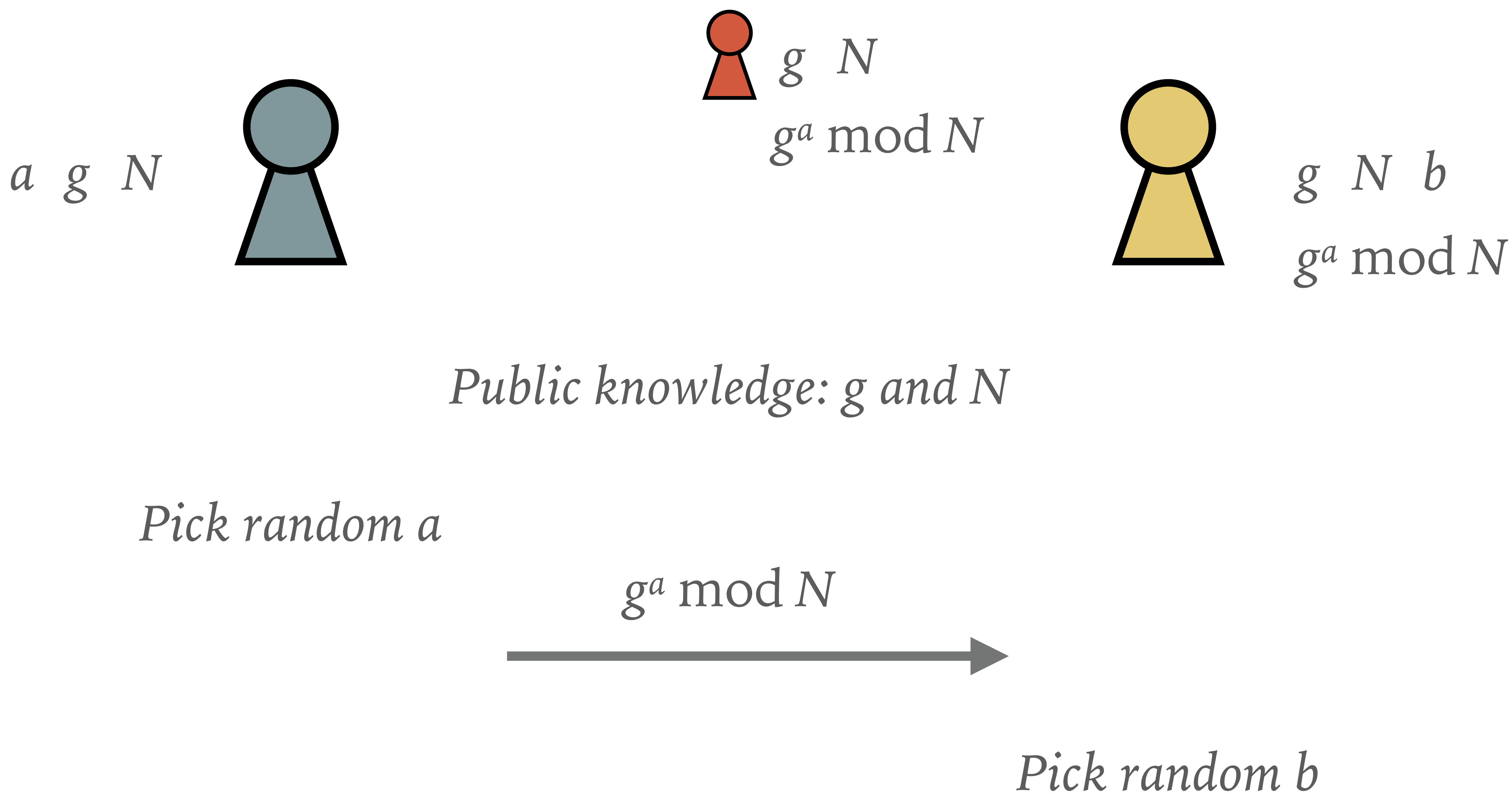
$g^a \bmod N$

Public knowledge: g and N

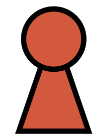
Pick random a

$g^a \bmod N$

DIFFIE-HELLMAN KEY EXCHANGE



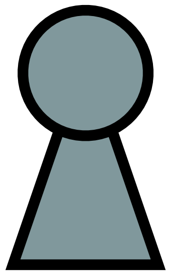
DIFFIE-HELLMAN KEY EXCHANGE

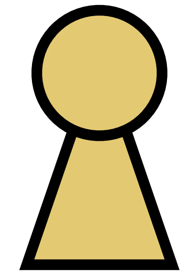


$g \quad N$

$g^a \bmod N$

$a \quad g \quad N$





$g \quad N \quad b$

$g^a \bmod N$

Public knowledge: g and N

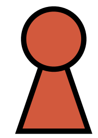
Pick random a

$g^a \bmod N$ →

Pick random b

$g^b \bmod N$ ←

DIFFIE-HELLMAN KEY EXCHANGE



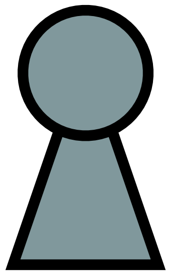
$g \quad N$

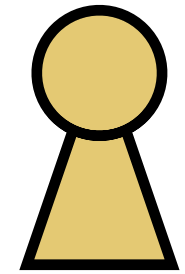
$g^a \bmod N$

$g^b \bmod N$

$a \quad g \quad N$

$g^b \bmod N$





$g \quad N \quad b$

$g^a \bmod N$

Public knowledge: g and N

Pick random a

$$\xrightarrow{g^a \bmod N}$$

Pick random b

$$\xleftarrow{g^b \bmod N}$$

DIFFIE-HELLMAN KEY EXCHANGE

$g \quad N$

$g^a \bmod N$

$g^b \bmod N$

$a \quad g \quad N$

$g^b \bmod N$

$g \quad N \quad b$

$g^a \bmod N$

Public knowledge: g and N

Pick random a

$$\xrightarrow{g^a \bmod N}$$

Pick random b

$$\xleftarrow{g^b \bmod N}$$

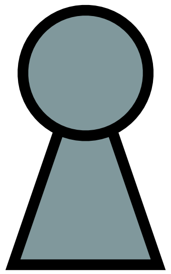
Compute $(g^b \bmod N)^a = g^{ab} \bmod N$

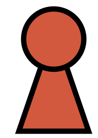
Compute $(g^a \bmod N)^b = g^{ab} \bmod N$

DIFFIE-HELLMAN KEY EXCHANGE

$a \quad g \quad N$

$g^b \bmod N$

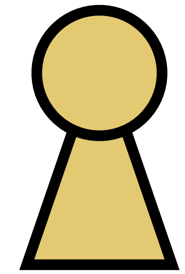




$g \quad N$

$g^a \bmod N$

$g^b \bmod N$



$g \quad N \quad b$

$g^a \bmod N$

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

DIFFIE-HELLMAN KEY EXCHANGE

$g \quad N$

$g^a \bmod N$

$g^b \bmod N$

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

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

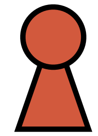
$$g^a \bmod N * g^b \bmod N \quad = g^{a+b} \bmod N$$

Key property:

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

But what about *active intermediaries*?

DIFFIE-HELLMAN KEY EXCHANGE



$g \quad N$

$g^a \bmod N$

$g^b \bmod N$

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

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:

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

Key property:

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

But what about *active intermediaries*?

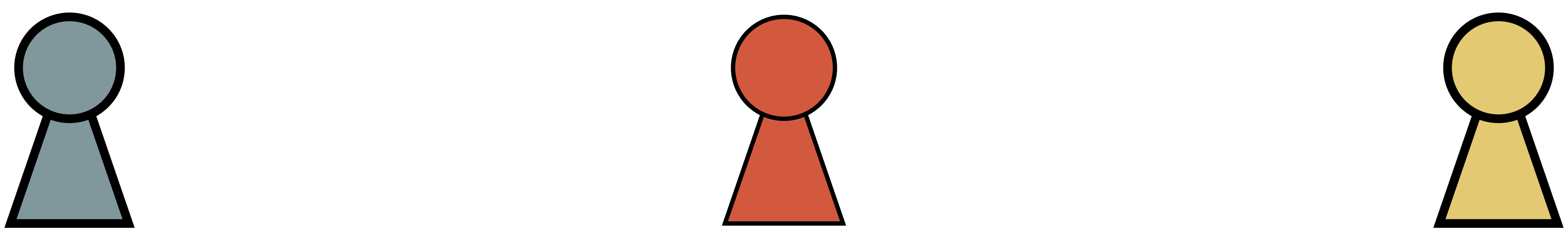
MAN-IN-THE-MIDDLE (MITM) ATTACKS

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

thinks he is talking to

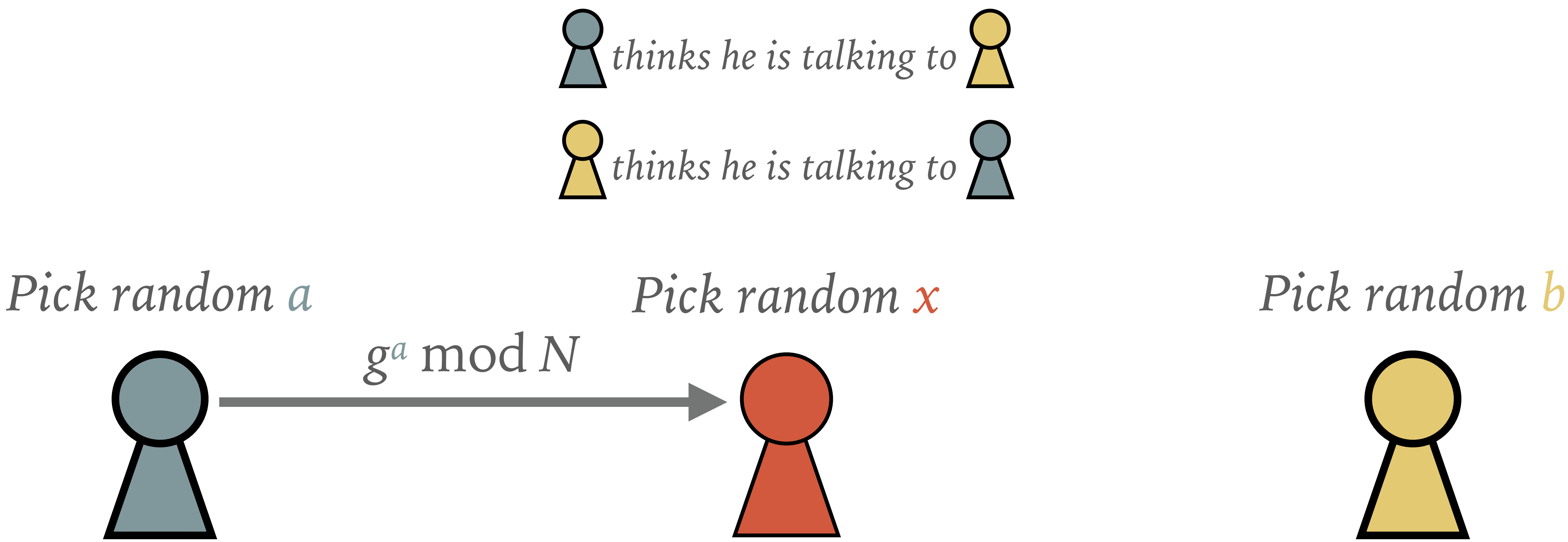
thinks he is talking to

Pick random a *Pick random x* *Pick random b*



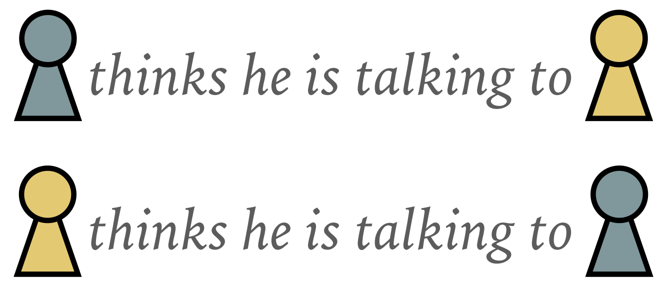
MAN-IN-THE-MIDDLE (MITM) ATTACKS

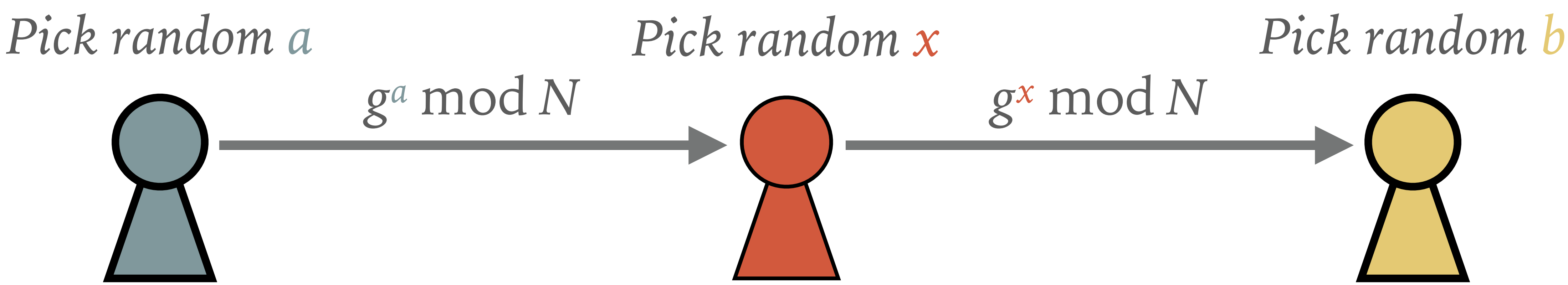
The attacker can interpose between the two communicating parties and insert, delete, and modify messages.



MAN-IN-THE-MIDDLE (MITM) ATTACKS

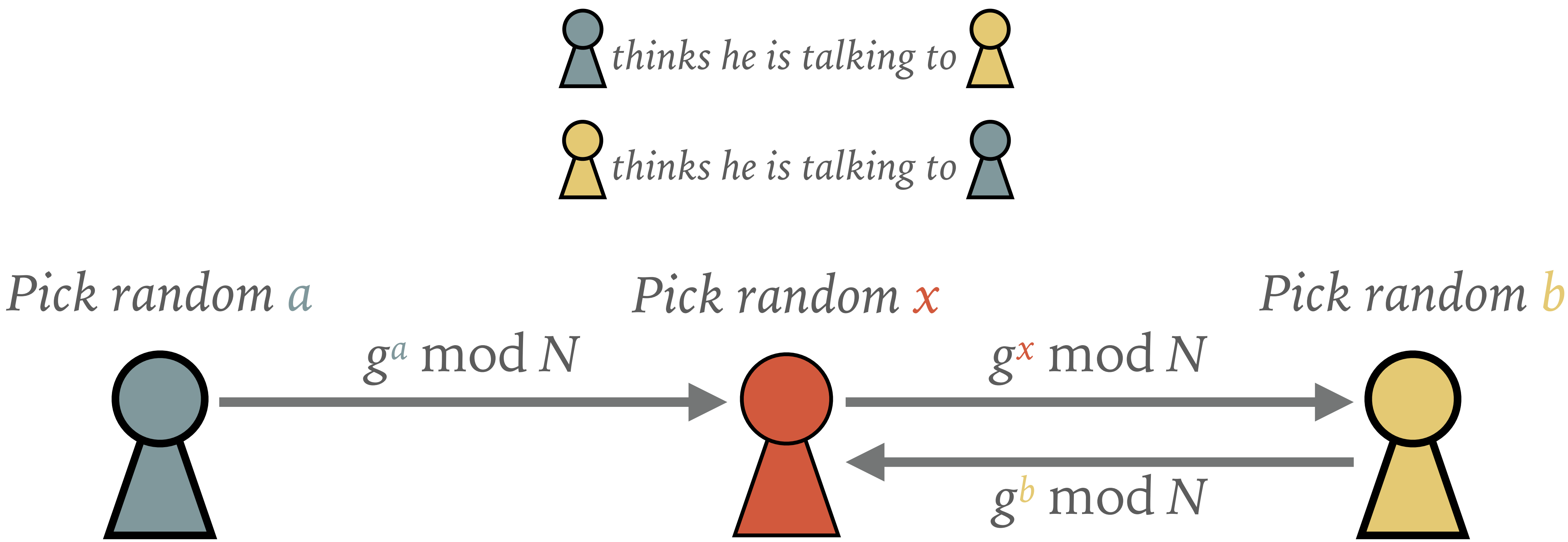
The attacker can interpose between the two communicating parties and insert, delete, and modify messages.





MAN-IN-THE-MIDDLE (MITM) ATTACKS

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

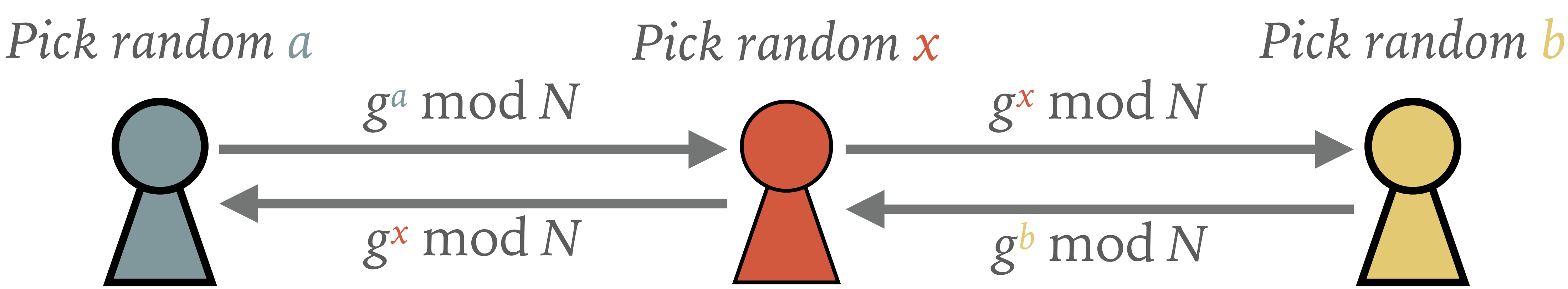


MAN-IN-THE-MIDDLE (MITM) ATTACKS

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

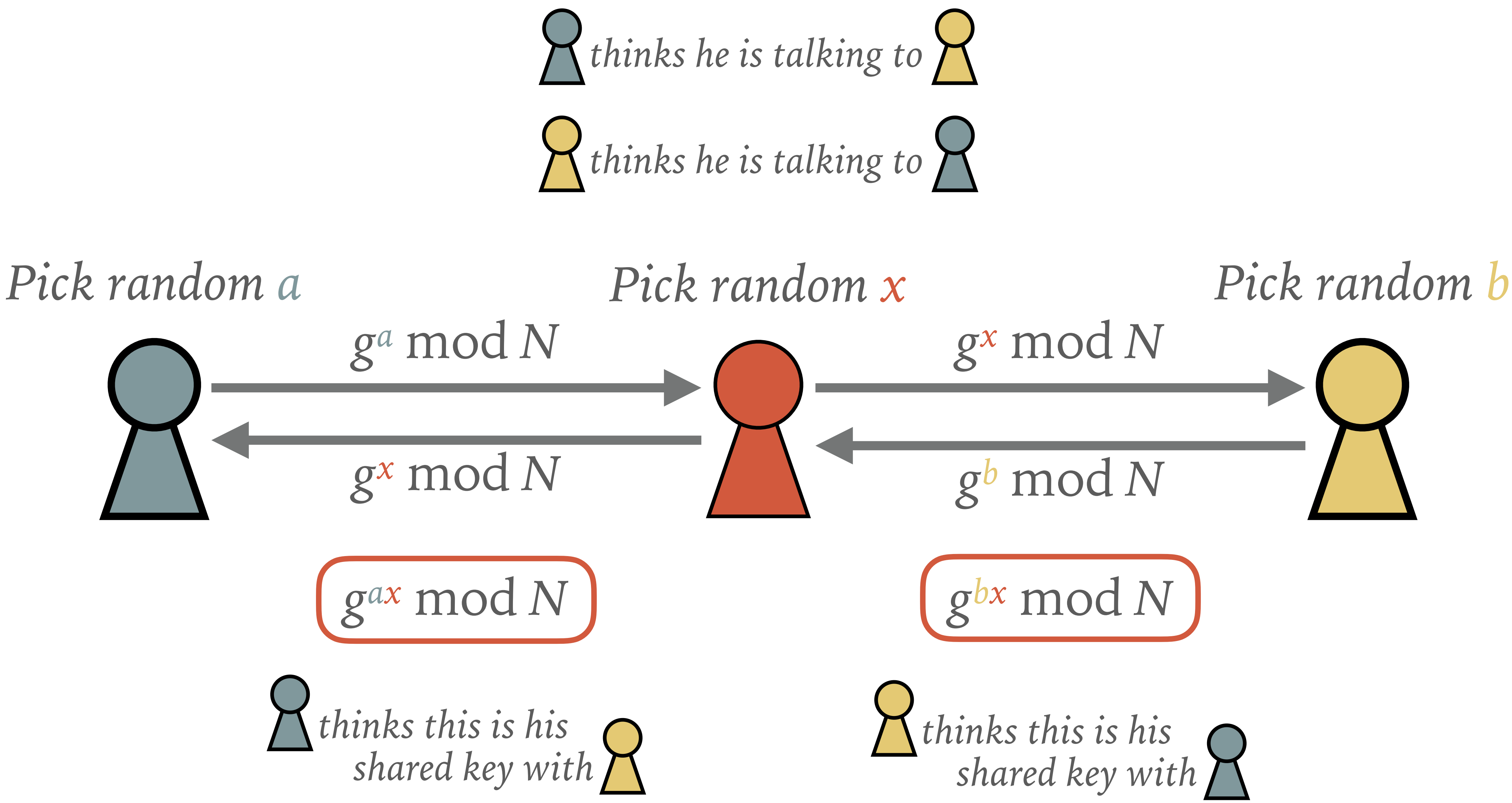
thinks he is talking to

thinks he is talking to



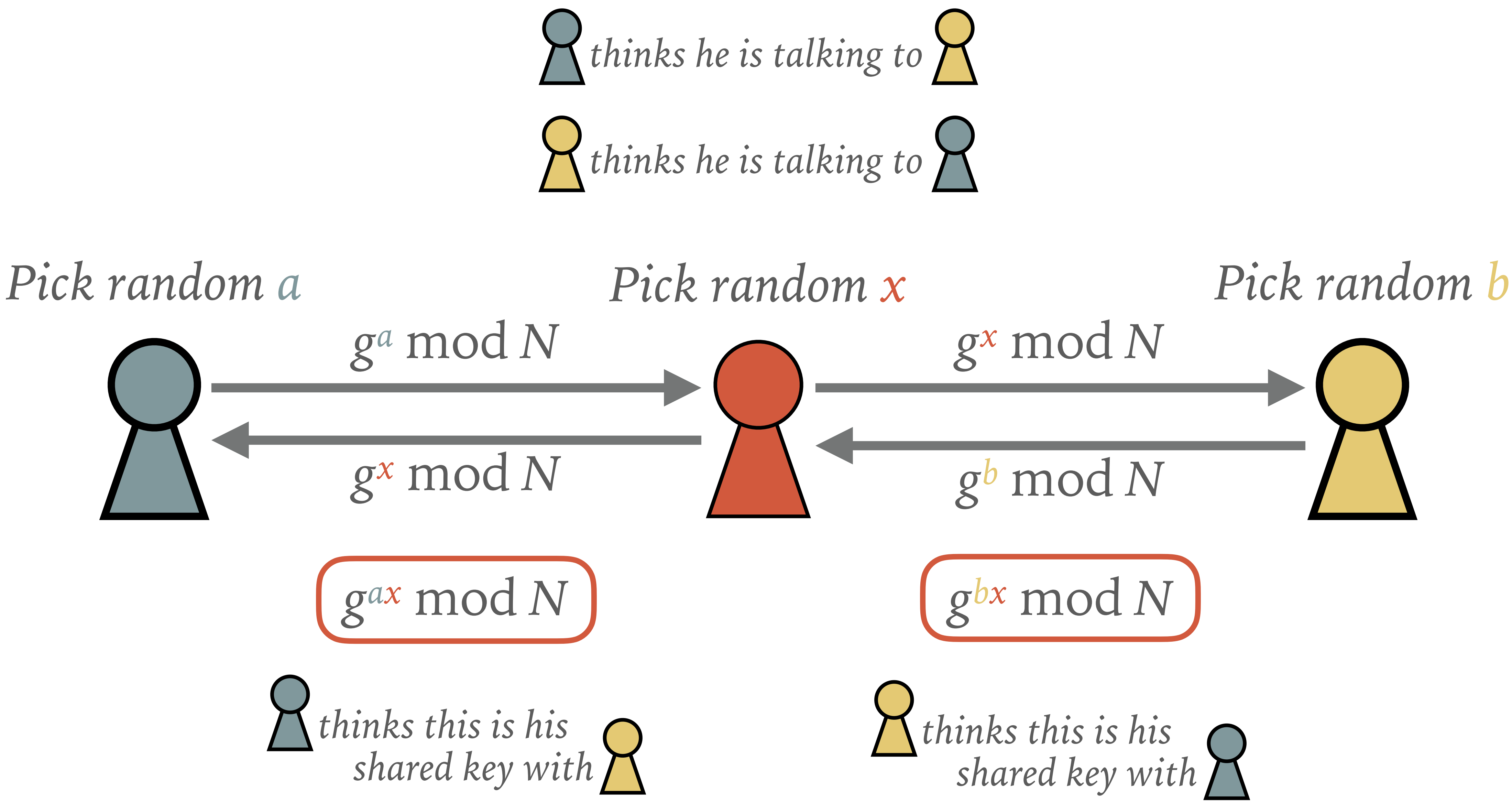
MAN-IN-THE-MIDDLE (MITM) ATTACKS

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



MAN-IN-THE-MIDDLE (MITM) ATTACKS

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



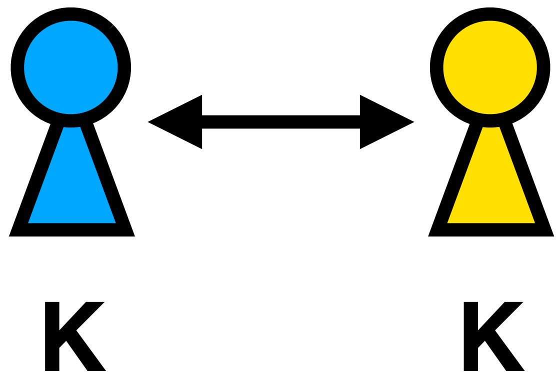
The attacker can now eavesdrop on the conversation.

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

TO FIX THIS PROBLEM WE NEED…

BLACKBOX #5:
PUBLIC KEY CRYPTOGRAPHY

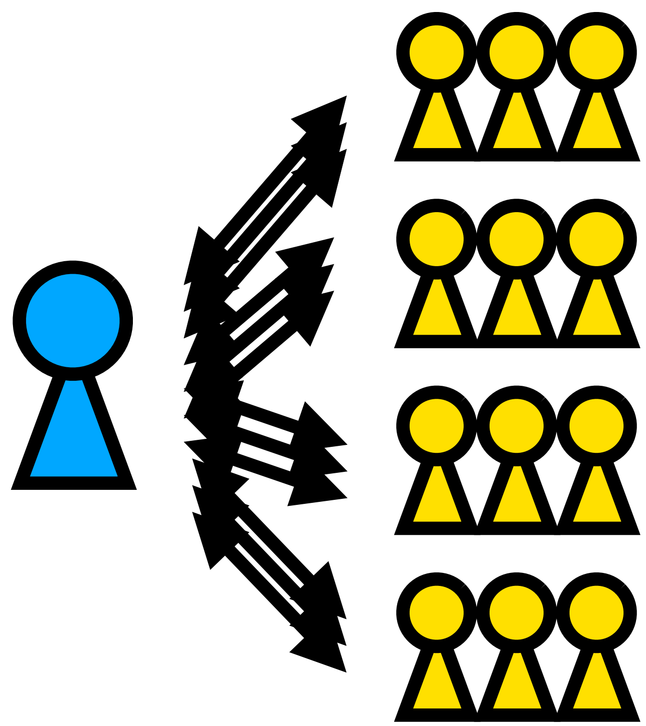
Shortcomings of symmetric key

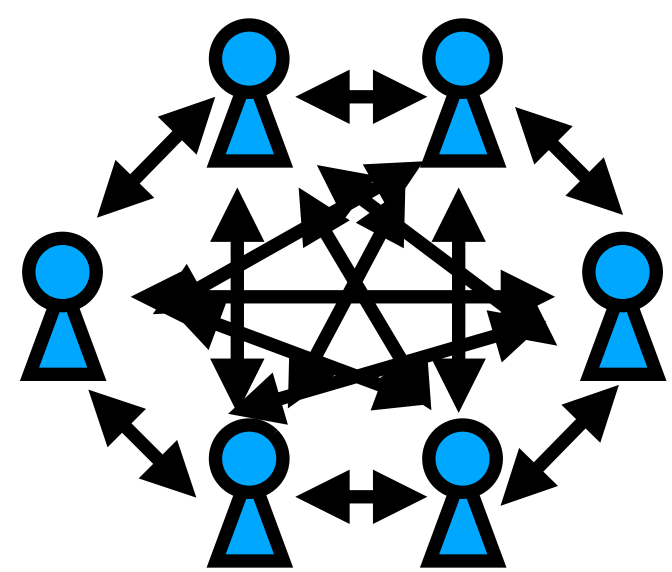


Establishing a pairwise key requires a **key exchange**, which requires both parties to be ***online***

Issue #1: Requires *pairwise* key exchanges

File downloads



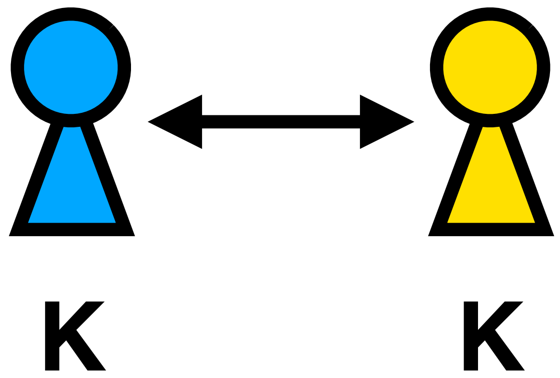


Email / chat

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

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

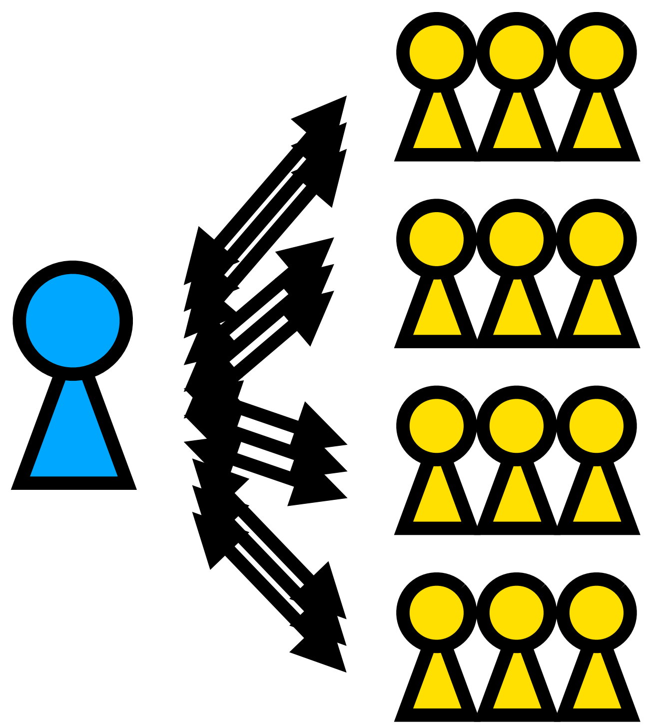
Shortcomings of symmetric key



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

Issue #2: Parties must be online

File downloads

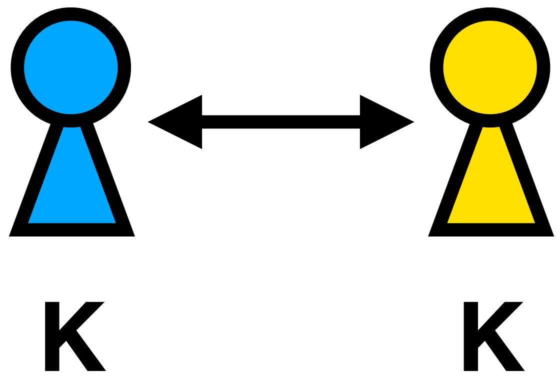


One-to-many:
O(N) key
exchanges

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?

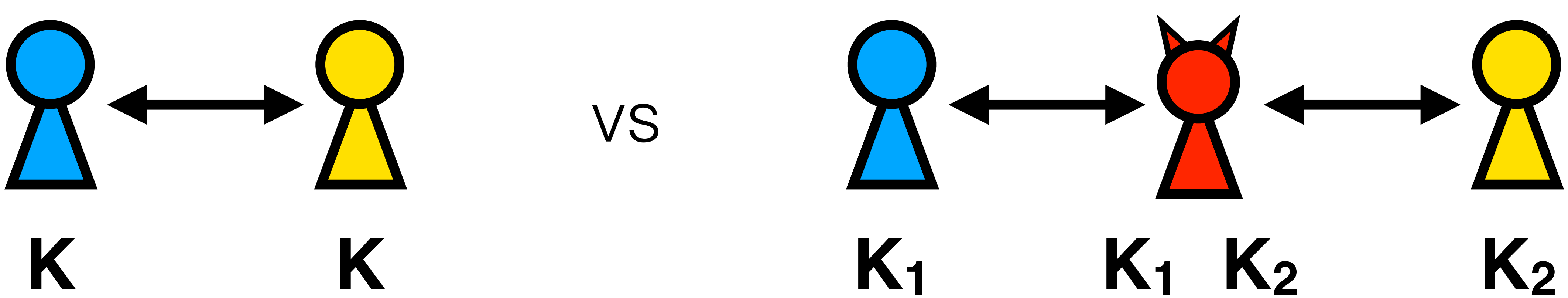
Shortcomings of symmetric key



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

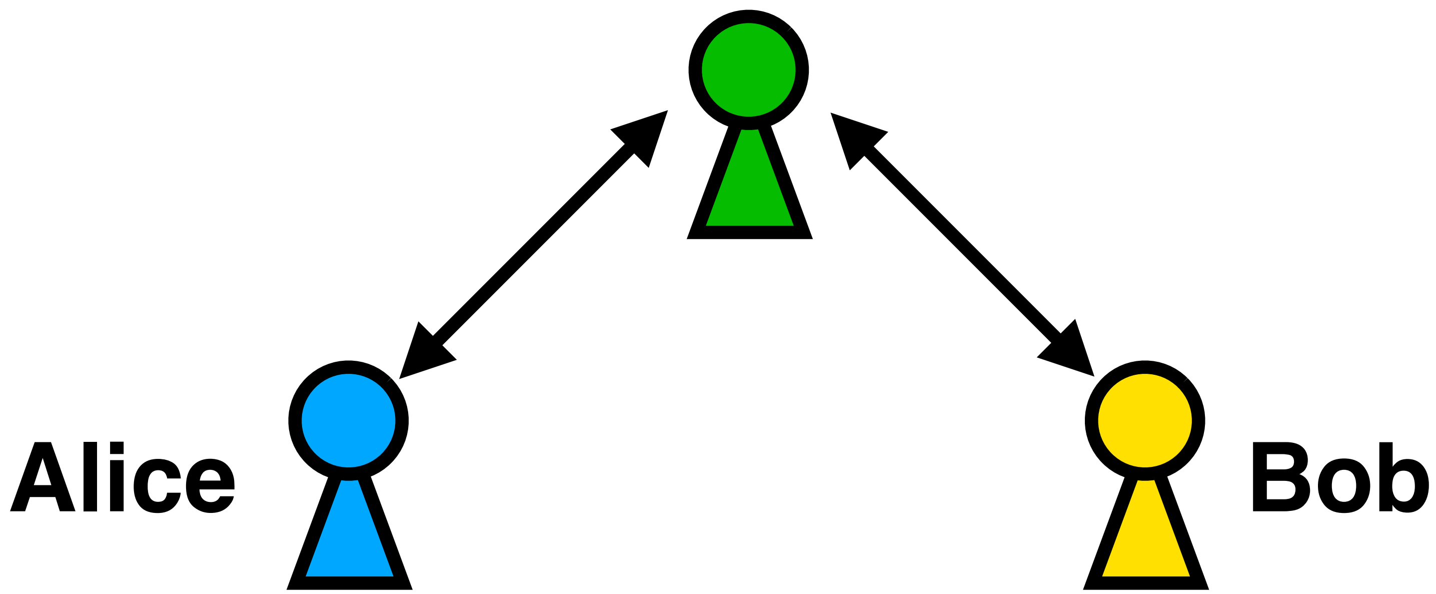
Issue #3: How do you know to whom you're talking?

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



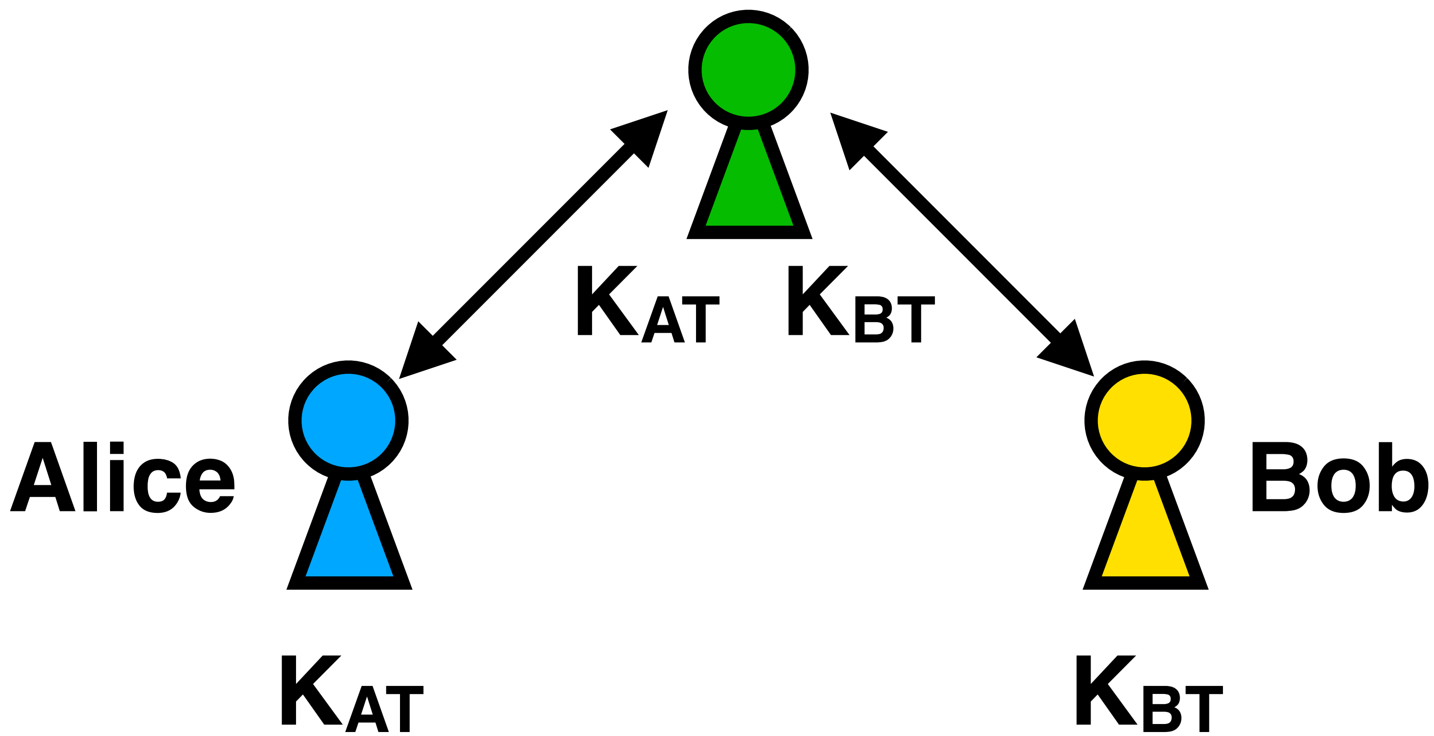
A protocol that solves this with ***trust***

Trent: A *trusted* third party



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

Trent: A *trusted* third party

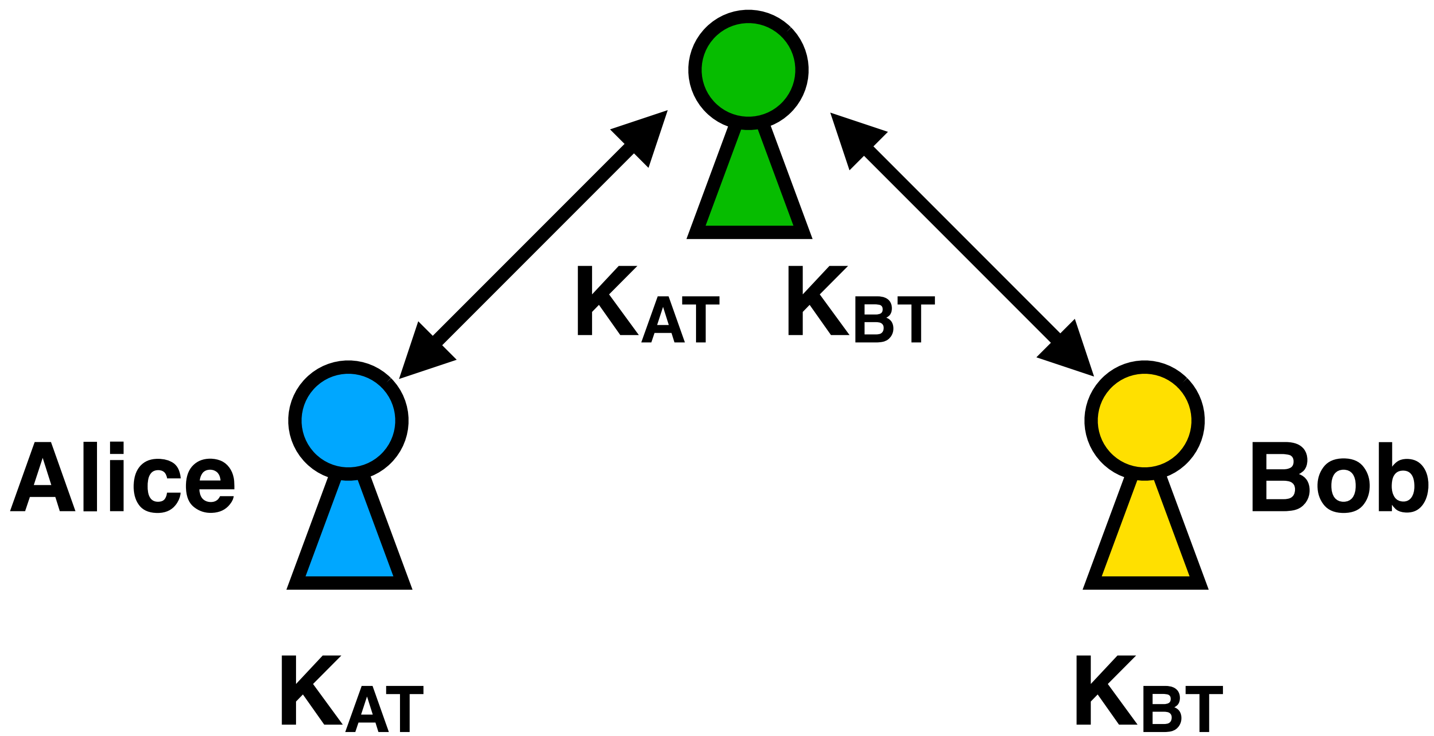


1. Everybody establishes a pairwise key with Trent

Good: *O(N) key exchanges*

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

Trent: A *trusted* third party

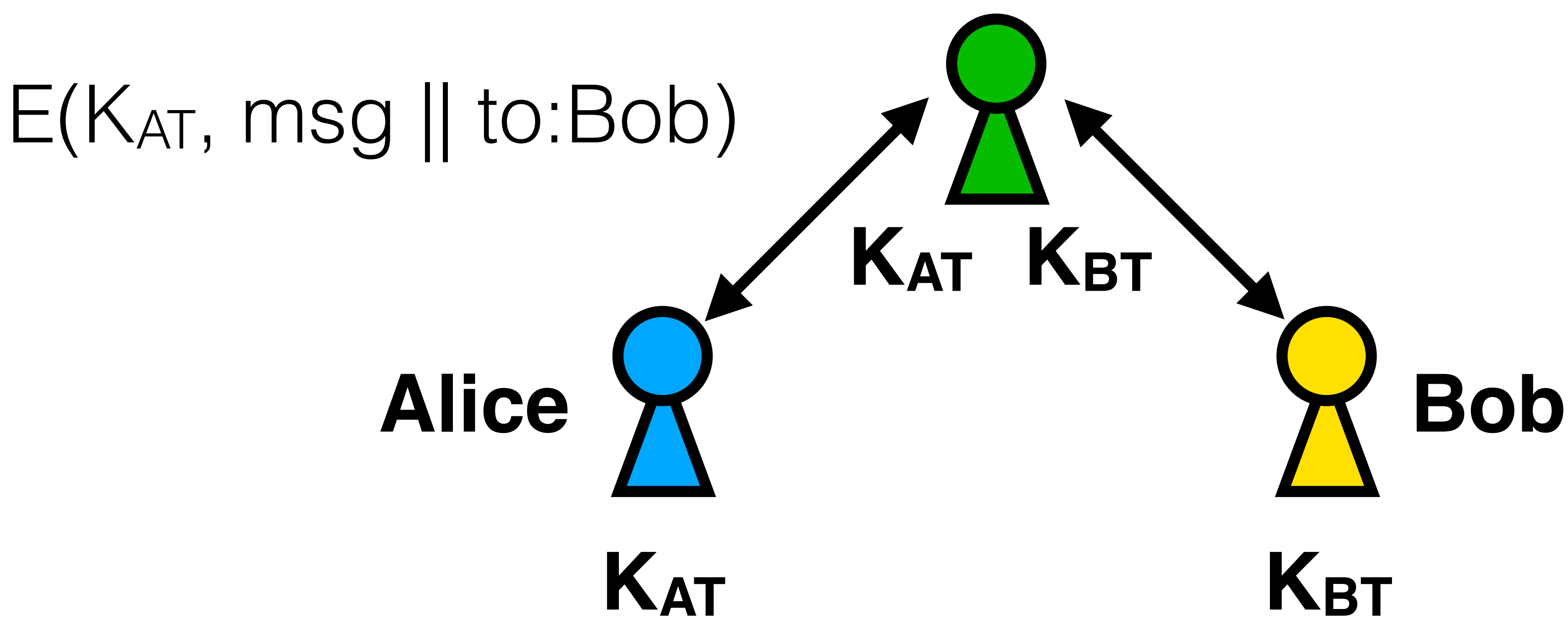


1. Everybody establishes a pairwise key with Trent
   Good: *O(N) key exchanges*

2. Trent validates each user's identity; includes in message
   Good: *Authenticated communication*

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

Trent: A *trusted* third party

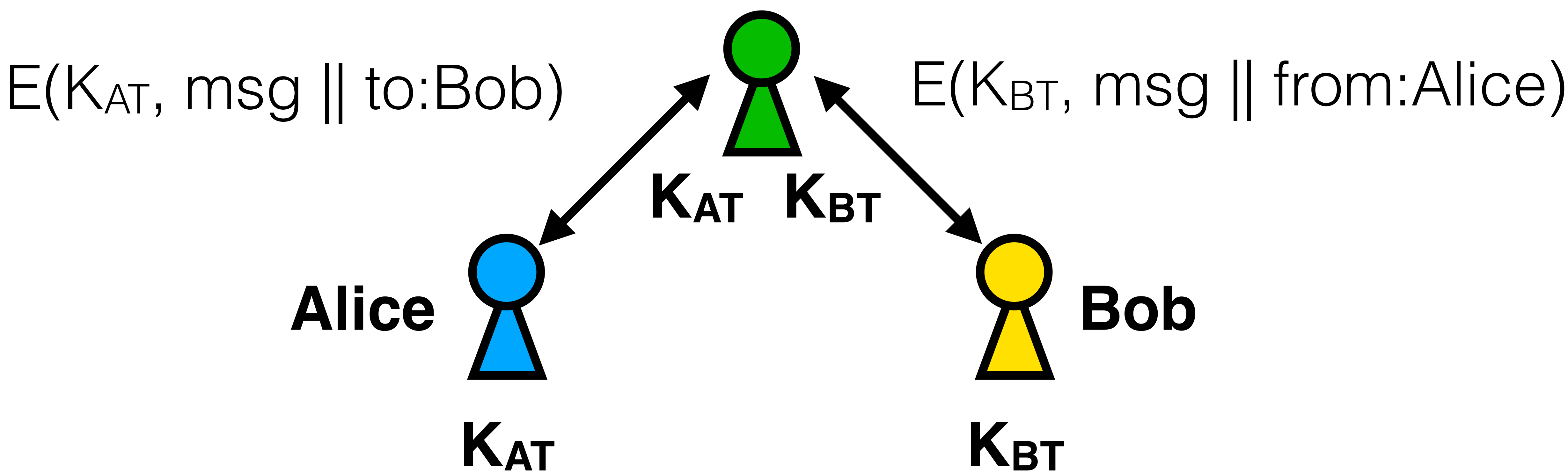


1. Everybody establishes a pairwise key with Trent
Good: *O(N) key exchanges*

2. Trent validates each user's identity; includes in message
Good: *Authenticated communication*

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

Trent: A *trusted* third party

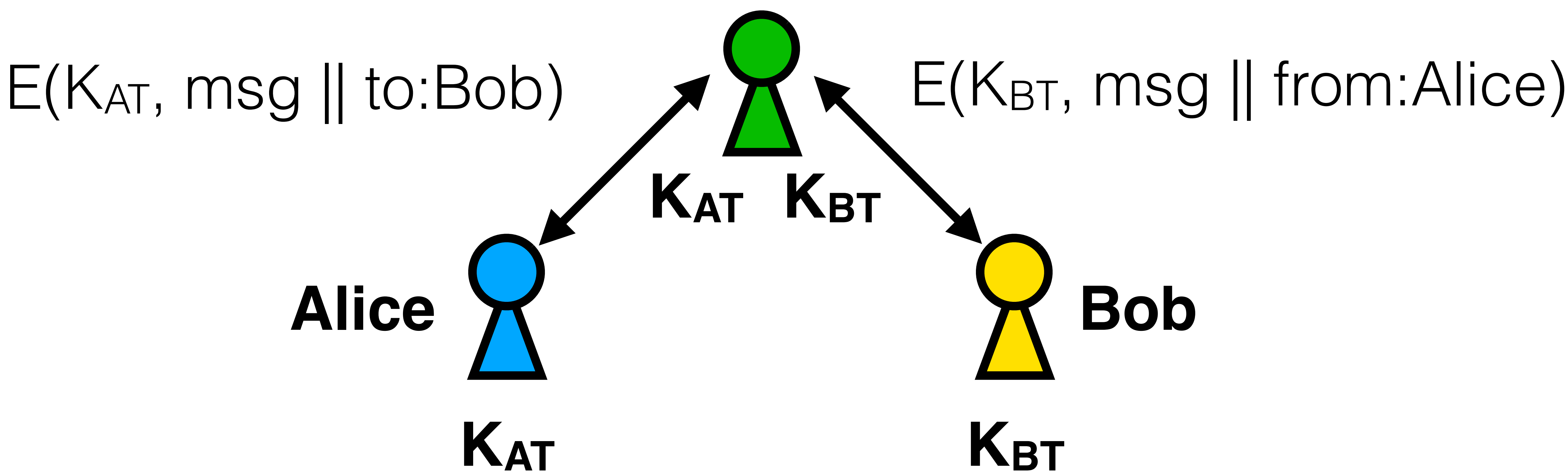


1. Everybody establishes a pairwise key with Trent
Good: *O(N) key exchanges*

2. Trent validates each user's identity; includes in message
Good: *Authenticated communication*

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

Trent: A *trusted* third party



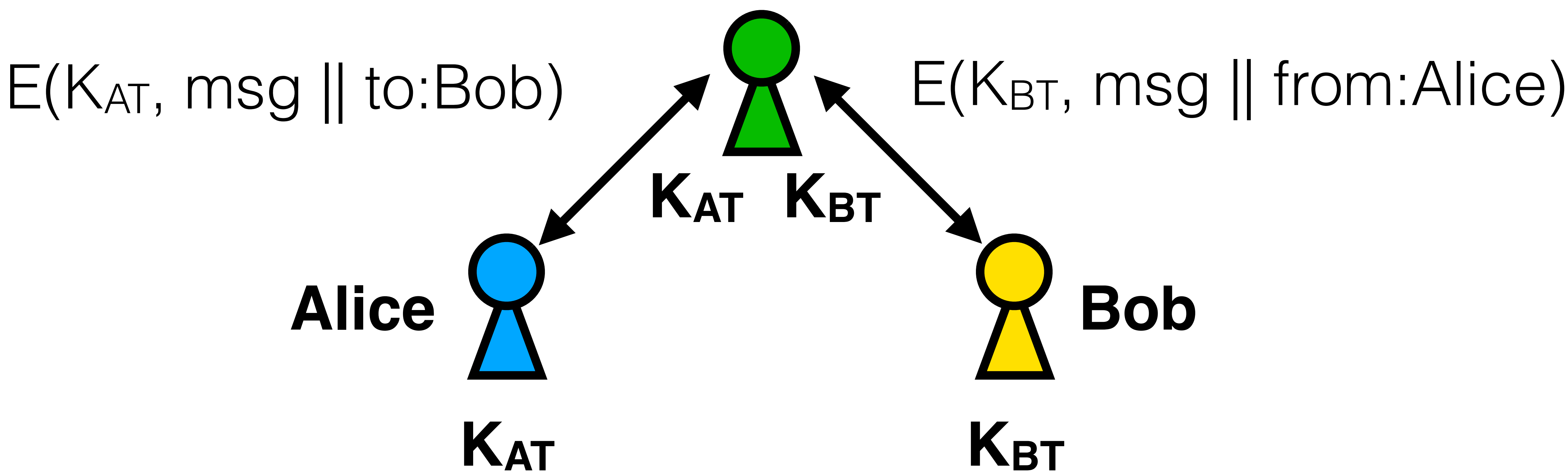
1. Everybody establishes a pairwise key with Trent
Good: *O(N) key exchanges*

2. Trent validates each user's identity; includes in message
Good: *Authenticated communication*

Bad: All messages get sent through Trent

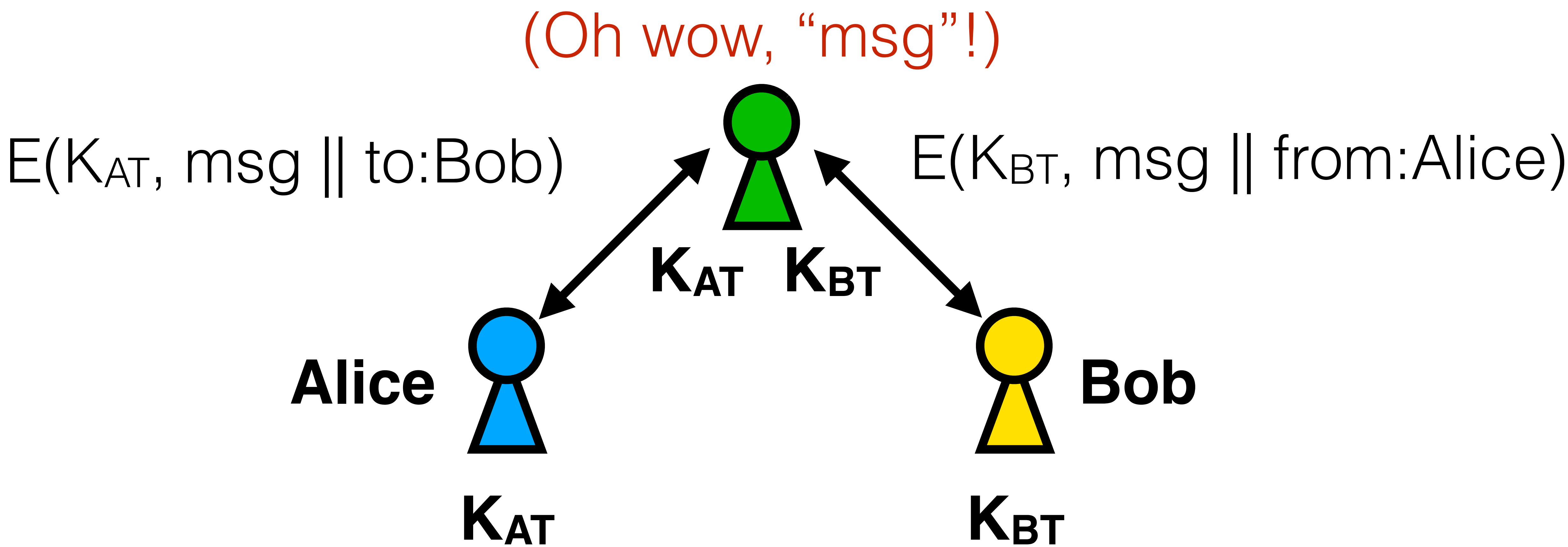
What are we trusting Trent not to do?

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



What are we trusting Trent not to do?

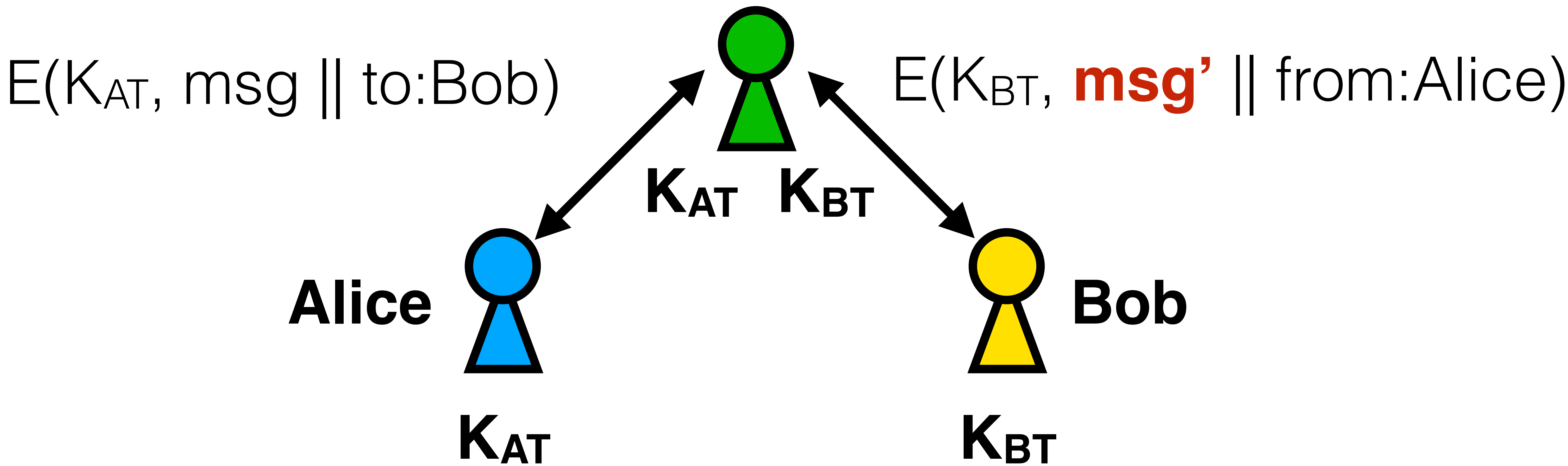
Just as "secure" meant nothing without an attack model, "trusted" means nothing without a **trust model**



1. Do not *read* messages

What are we trusting Trent not to do?

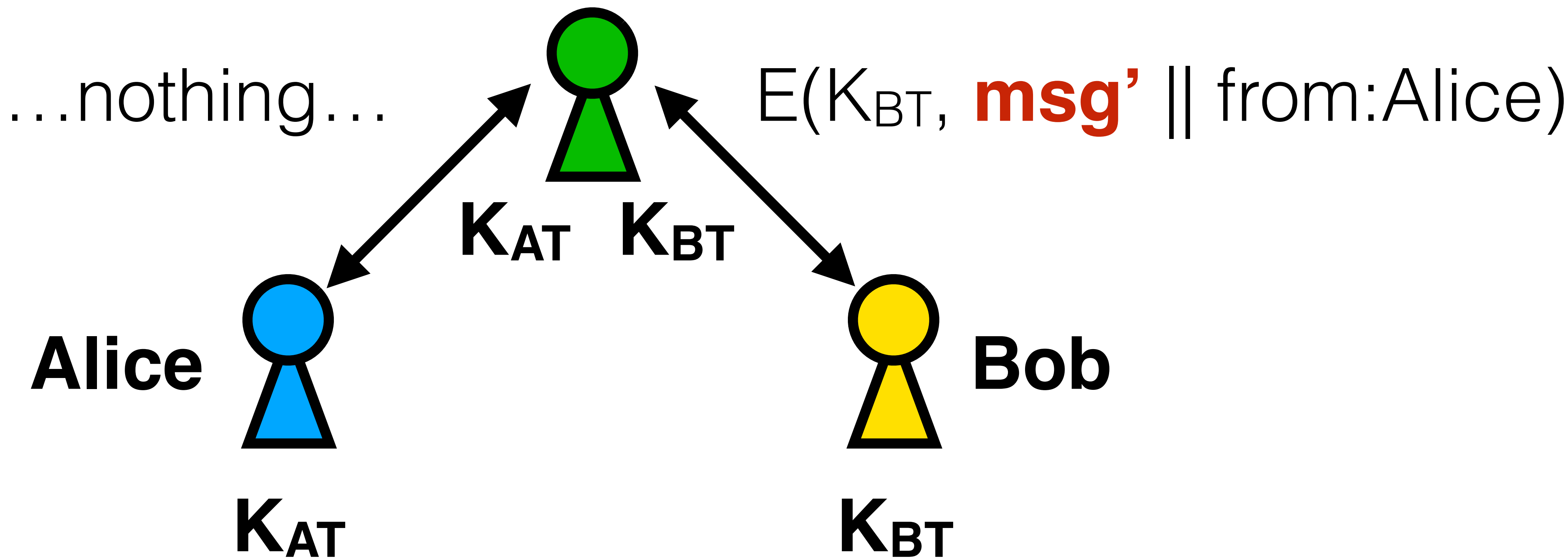
Just as "secure" meant nothing without an attack model, "trusted" means nothing without a **trust model**



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

What are we trusting Trent not to do?

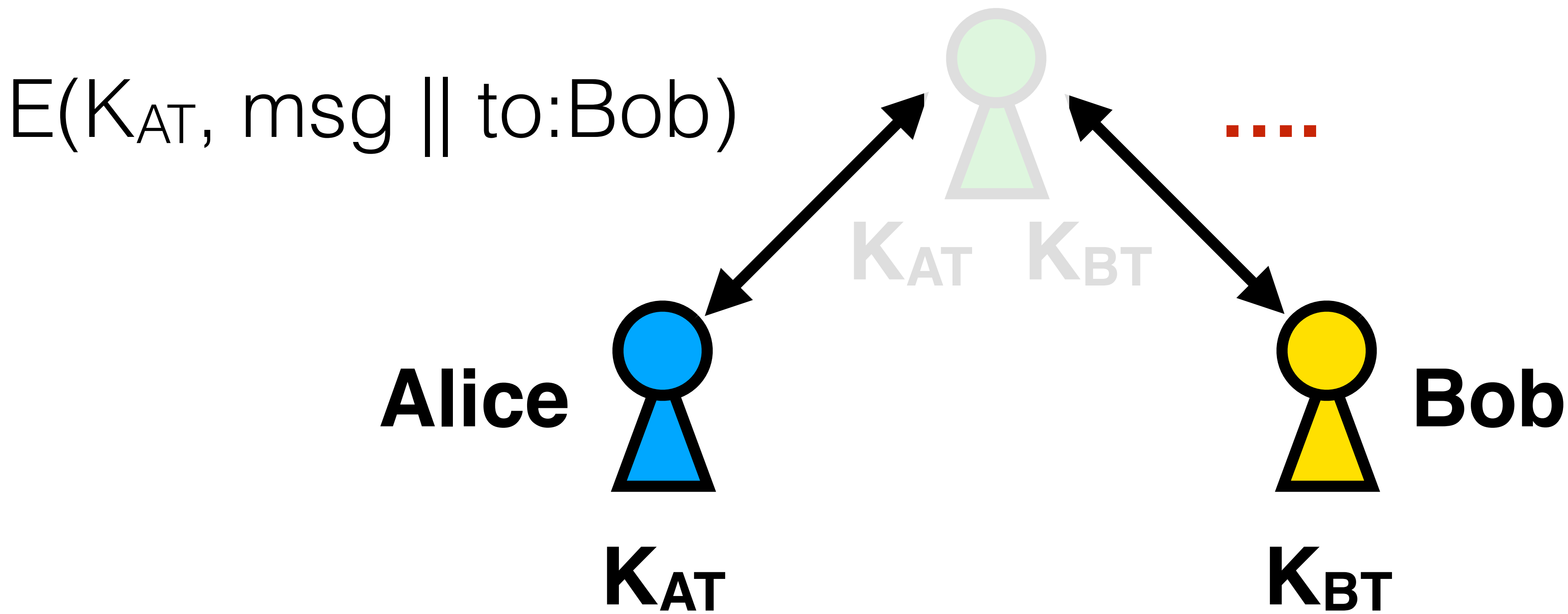
Just as “secure” meant nothing without an attack model, “trusted” means nothing without a **trust model**



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**

- Inputs
  - Source of randomness
  - Maximum key length L
- Outputs: a *key pair*
  - *PK* = **public key**
  - *SK* = **secret key**

This is a *randomized* algorithm
(nondeterministic output)

Difficult to infer SK from PK
Only one person should know SK;
PK should be public to *all*

PK and SK are intrinsically bound together:
for a given PK, there is a single *corresponding* SK

Example: RSA's public keys are a pair: (exponent, modulus)

Public key encryption

A public key encryption scheme comprises three algorithms

Encryption E(PK, msg)

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

This is a *randomized* algorithm
(vanilla RSA is deterministic; in practice, RSA-PKCS is used instead, which adds a nonce to the message)

PK a.k.a. "Encryption key"

Anyone who knows Alice's PK can encrypt a message to her…

Public key encryption

A public key encryption scheme comprises three algorithms

Decryption **D(SK, c)**

- Inputs
  - **Secret** key SK
  - Cipher text c
- Outputs: original msg

This is a *deterministic* algorithm
Should always return the original message

…but only Alice can decrypt that message

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)

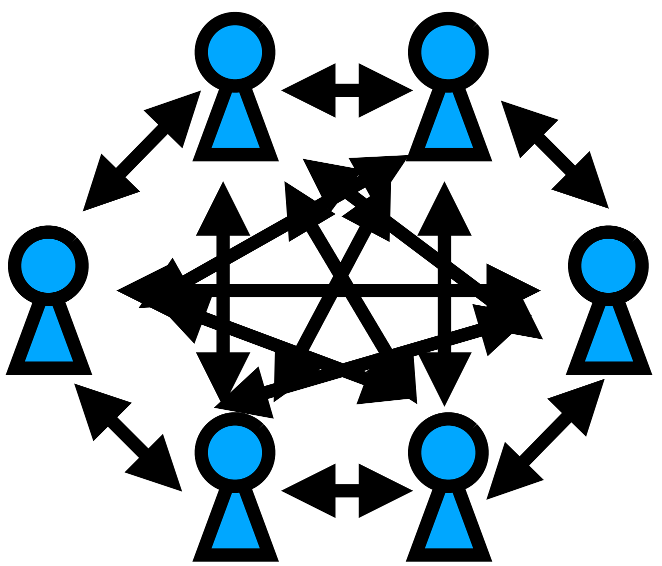
E() should approximate a one-way trapdoor function: cannot invert without access to SK

Protocols with public key encryption

Goal: deliver a confidential message

Symmetric key

Email / chat



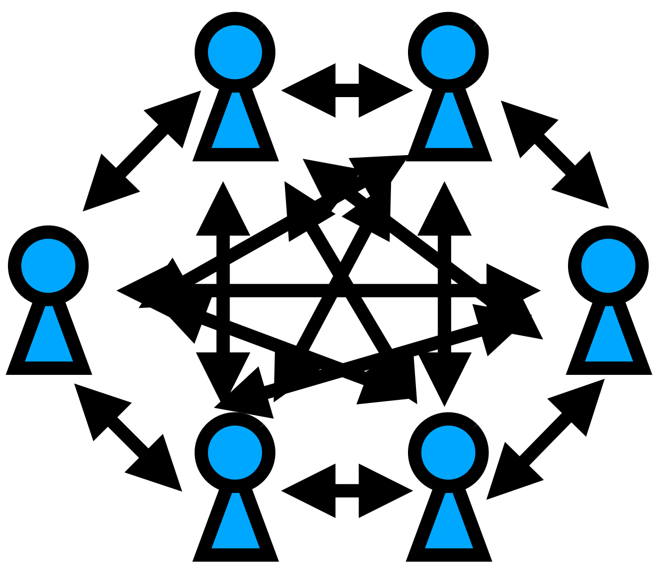
All-to-all:
O(N^2) key
exchanges

Protocols with public key encryption

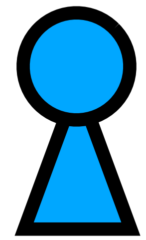
Goal: deliver a confidential message

Symmetric key

Email / chat



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



Generate public/private
key pair (PK,SK)

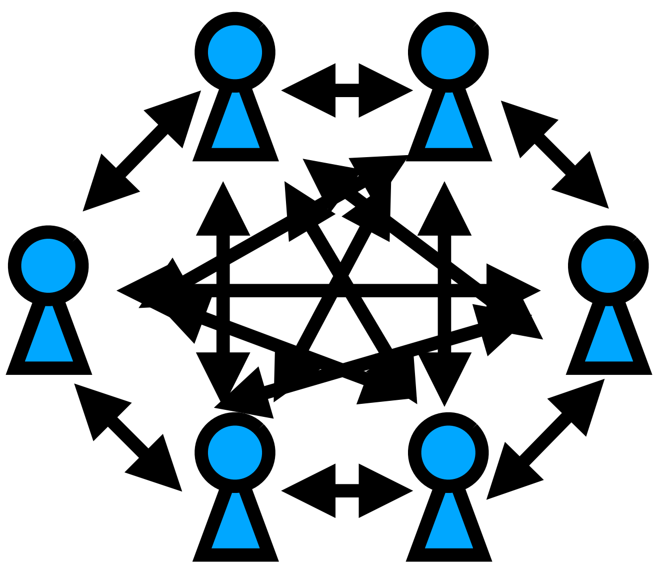
Annouce PK publicly
(on website, in newspaper, …)

Protocols with public key encryption

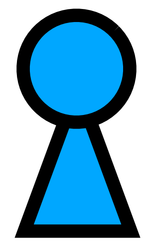
Goal: deliver a confidential message

Symmetric key

Email / chat

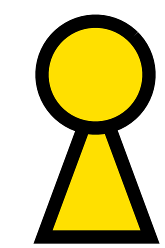


All-to-all: $O(N^2)$ key exchanges



Generate public/private key pair (PK,SK)

Annouce PK publicly (on website, in newspaper, …)



Obtain PK

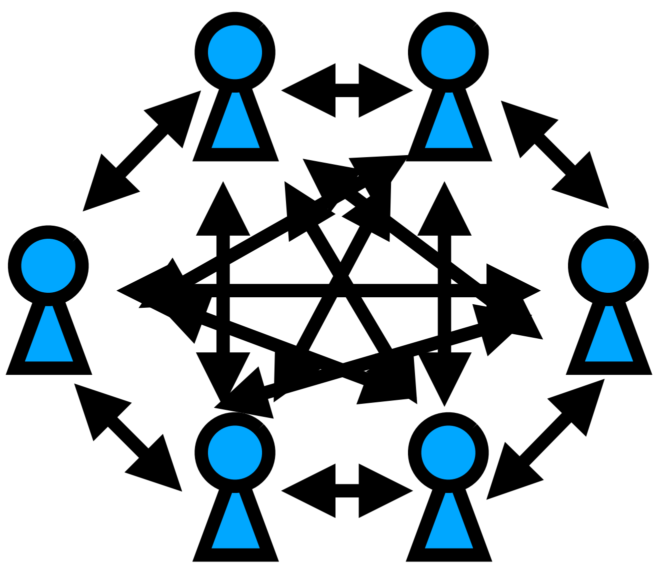
Send c = E(PK, msg)

Protocols with public key encryption

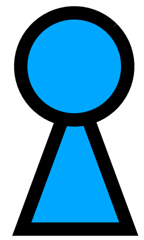
Goal: deliver a confidential message

Symmetric key

Email / chat

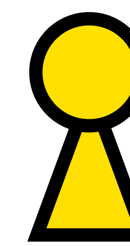


All-to-all:
$O(N^2)$ key exchanges



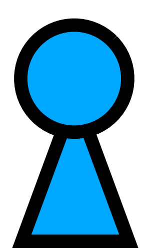
Generate public/private key pair (PK,SK)

Annouce PK publicly
(on website, in newspaper, …)



Obtain PK

Send c = E(PK, msg)



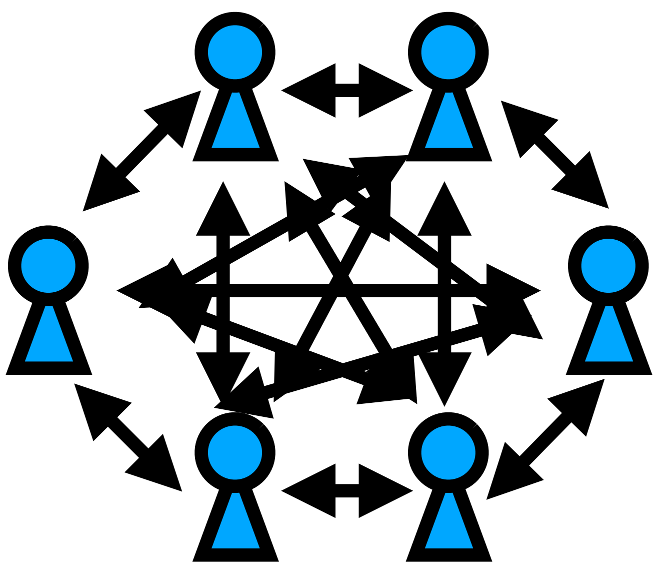
Decrypt D(SK, c) = msg

Protocols with public key encryption

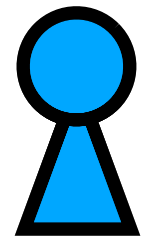
Goal: deliver a confidential message

Symmetric key

Email / chat

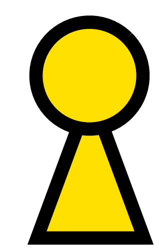


All-to-all:
$O(N^2)$ key exchanges



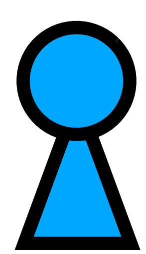
Generate public/private key pair (PK,SK)

Annouce PK publicly (on website, in newspaper, …)



Obtain PK

Send c = E(PK, msg)



Decrypt D(SK, c) = msg

O(N) keys in total

Overcoming fixed message sizes

Encryption **E(PK, 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

Overcoming fixed message sizes

Encryption **E(PK, 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 *slooooow!*

Overcoming fixed message sizes

Encryption E(PK, msg)

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

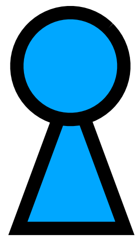
Like block ciphers, but there are not "modes" of public key encryption

Public key operations are *slooooow!*

Symmetric key operations are fast

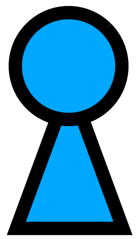
Hybrid encryption

Hybrid encryption

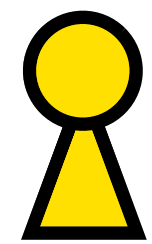


Generate public/private key pair (PK,SK); publicize PK

Hybrid encryption



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



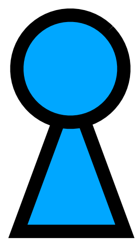
Obtain PK

Generate *symmetric* key K

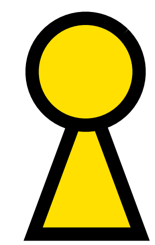
Compute c_{msg} = e(K, msg)

Compute c_K = E(PK, K)

Hybrid encryption



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



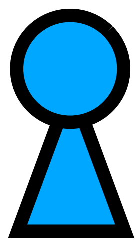
Obtain PK

Generate *symmetric* key K

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

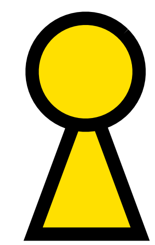
Public key Compute c_K = E(PK, K)

Hybrid encryption



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

Obtain PK

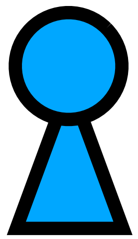


Generate *symmetric* key K

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

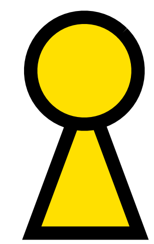
Public key Compute c_K = E(PK, K) ***Now throw away K***

Hybrid encryption



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

Obtain PK



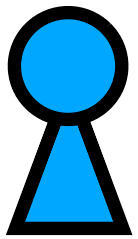
Generate *symmetric* key K

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

Public key Compute $c_K = E(PK, K)$ ***Now throw away K***

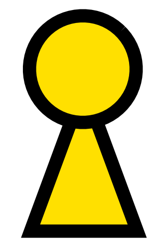
Send $c_K \,||\, c_{msg}$

Hybrid encryption



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

Obtain PK

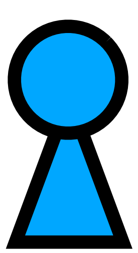


Generate *symmetric* key K

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

Public key Compute $c_K = E(PK, K)$ ***Now throw away K***

Send $c_K \,||\, c_{msg}$



Decrypt $D(SK, c_K) = K$

Decrypt $d(K, c_{msg}) = msg$

Hybrid encryption

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

Obtain PK

Generate *symmetric* key K

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

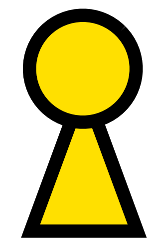
Public key Compute c_K = E(PK, K) ***Now throw away K***

Send c_K || c_{msg}

Decrypt D(SK, c_K) = K *Public key*

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

Hybrid encryption



Obtain PK

Generate *symmetric* key K

Compute c_{msg} = e(K, msg)

Compute c_K = E(PK, K)

Send $c_K \,||\, c_{msg}$

The easy key distribution of public key

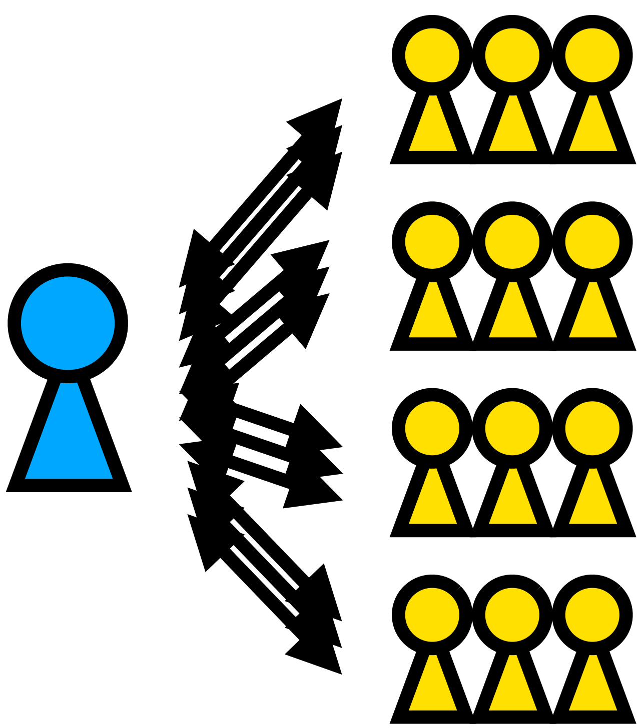
The speed and arbitrary message length of symmetric key

Protocols with public key cryptography

Goal: determine from whom a message came

Symmetric key

File downloads



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

Ideally, a user (blue) could post a message (e.g., sensitive documents or a kernel update), and then go offline

And downloaders (yellow) could subsequently infer the message's authenticity without having to have already established a pairwise key with the publisher

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*

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)

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”

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

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)

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

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)) = Yes

Security

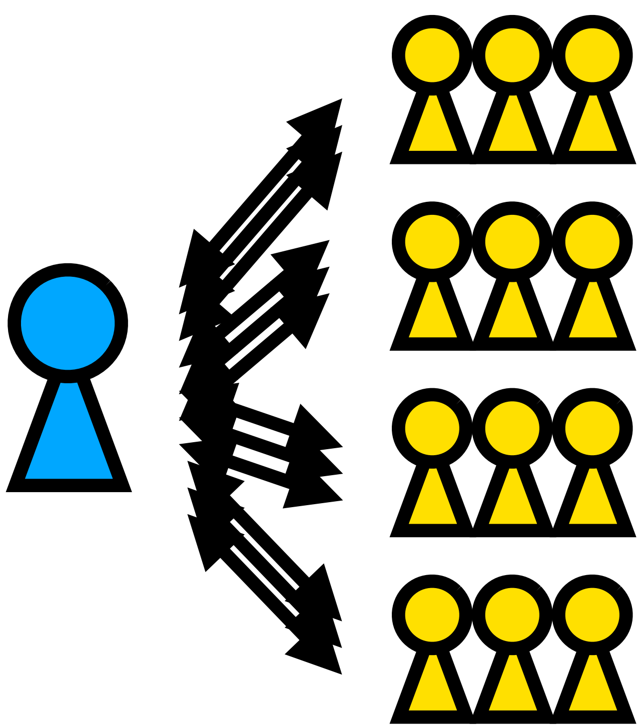
Same as with MACs: even after a chosen plaintext attack, the attacker cannot demonstrate an existential forgery

Protocols with digital signatures

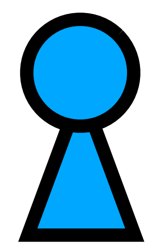
Goal: determine from whom a message came

Symmetric key

File downloads



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



Generate public/private key pair (PK,SK)

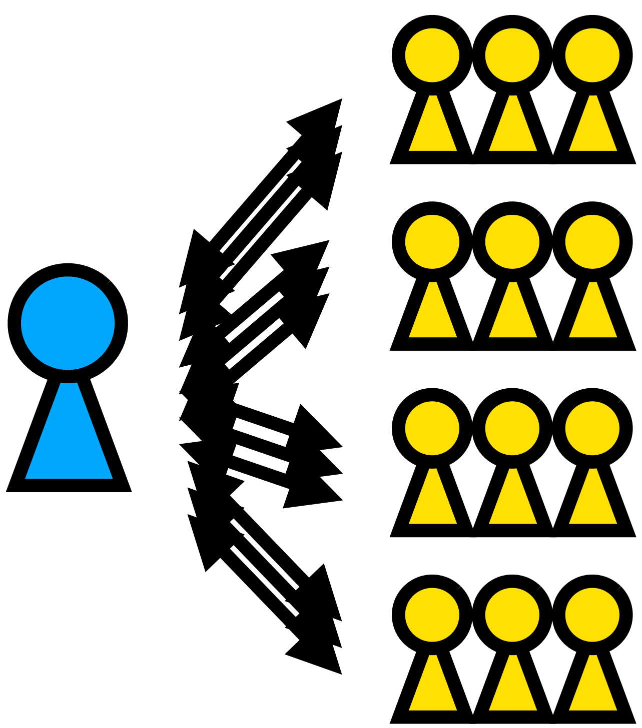
Annouce PK publicly
(on website, in newspaper, …)

Protocols with digital signatures

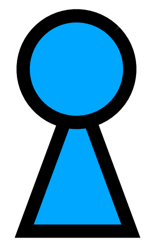
Goal: determine from whom a message came

Symmetric key

File downloads



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



Generate public/private key pair (PK,SK)

Annouce PK publicly
(on website, in newspaper, …)

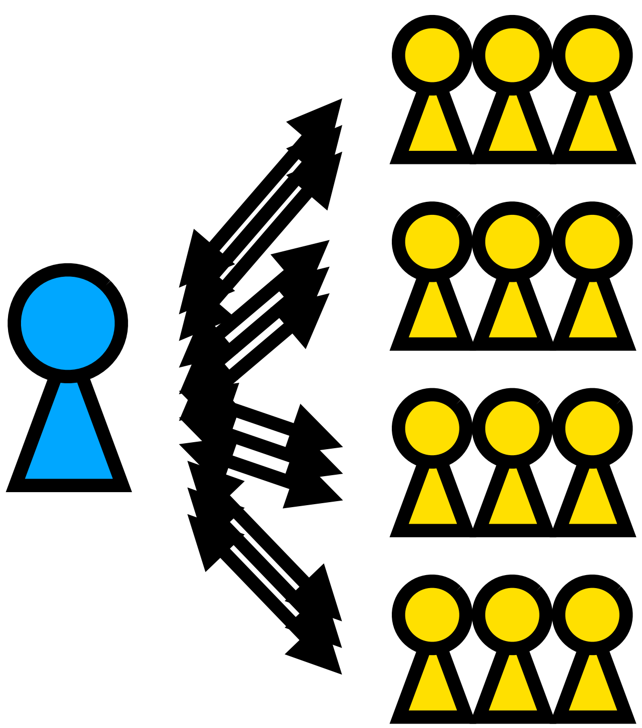
Compute sig = Sgn(SK, msg)

Protocols with digital signatures

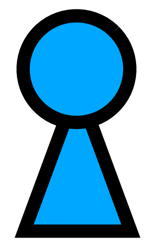
Goal: determine from whom a message came

Symmetric key

File downloads



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



Generate public/private
key pair (PK,SK)

Annouce PK publicly
(on website, in newspaper, …)

Compute sig = Sgn(SK, msg)

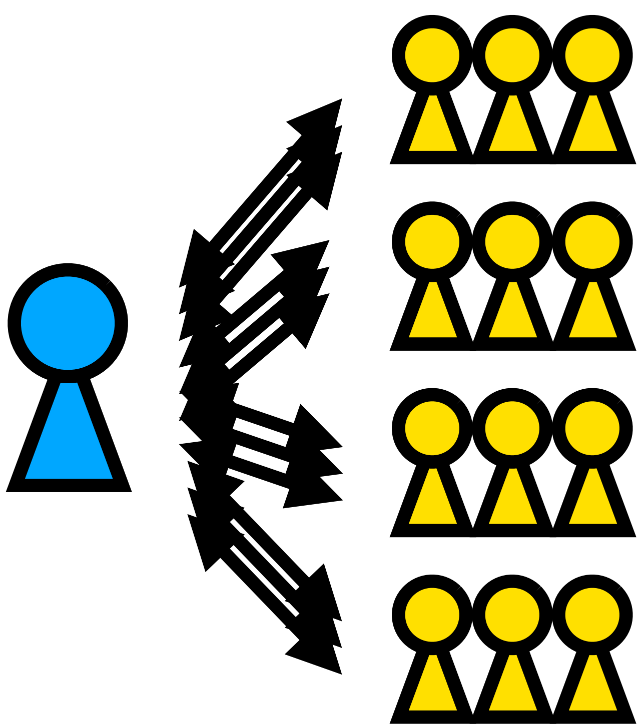
Publish msg || sig

Protocols with digital signatures

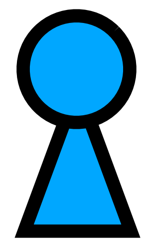
Goal: determine from whom a message came

Symmetric key

File downloads



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



Generate public/private
key pair (PK,SK)

Annouce PK publicly
(on website, in newspaper, …)

Compute sig = Sgn(SK, msg)

Publish msg || sig

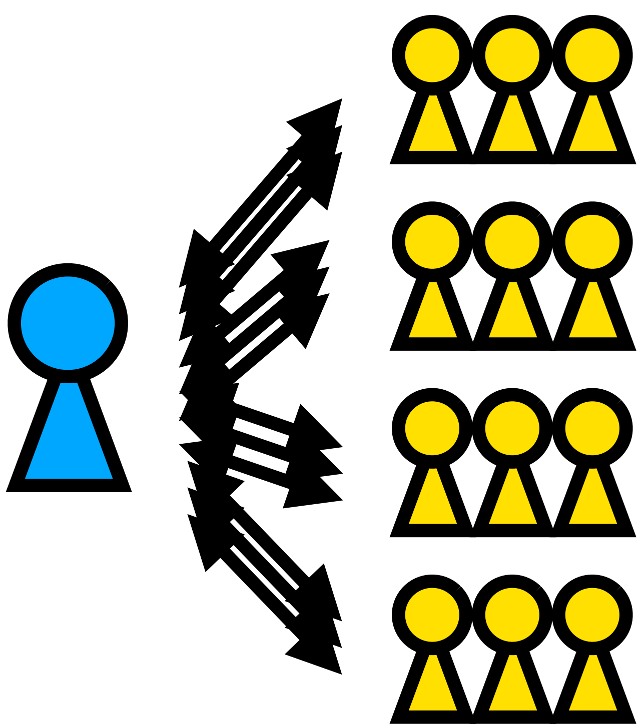
can now go offline!

Protocols with digital signatures

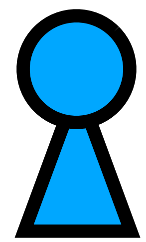
Goal: determine from whom a message came

Symmetric key

File downloads



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



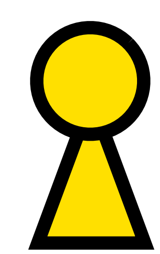
Generate public/private
key pair (PK,SK)

Annouce PK publicly
(on website, in newspaper, …)

Compute sig = Sgn(SK, msg)

Publish msg || sig

can now go offline!



Obtain PK, msg || sig

Vfy(PK, msg, sig)

Digital signature properties

Digital signature properties

Authenticity

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

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

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

Do *handwritten* signatures at the end of a letter have these 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

Do *handwritten* signatures at the end of a letter have these properties?

Authenticity

Would require unforgeable handwritten signatures. This is the one property they *sort of* get

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

Do *handwritten* signatures at the end of a letter have these properties?

Authenticity

Would require unforgeable handwritten signatures. This is the one property they *sort of* get

Integrity

Would require having a signature that depended on each part in the body of the letter

Non-repudiation

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

Do *handwritten* signatures at the end of a letter have these properties?

Authenticity	Would require unforgeable handwritten signatures. This is the one property they *sort of* get
Integrity	Would require having a signature that depended on each part in the body of the letter
Non-repudiation	Would require both of the above (unforgeable signature that depends on each part of letter)