CS 88: Security and Privacy

17: Asymmetric Key Cryptography and PKI

03-28-2024

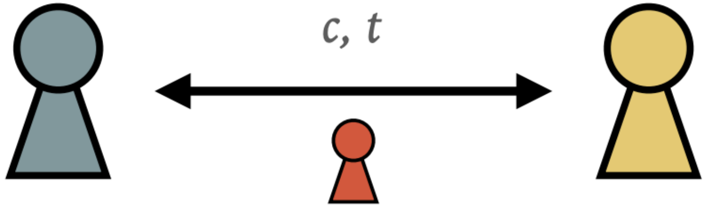
slides adapted from Dave Levine

SWARTHMORE COLLEGE

BLACKBOX #3:
HASH FUNCTIONS

Authenticated Encryption: Secrecy + Integrity

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



`k1, k2`

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

`k1, k2`

```
Dec(k1, c) = m
```

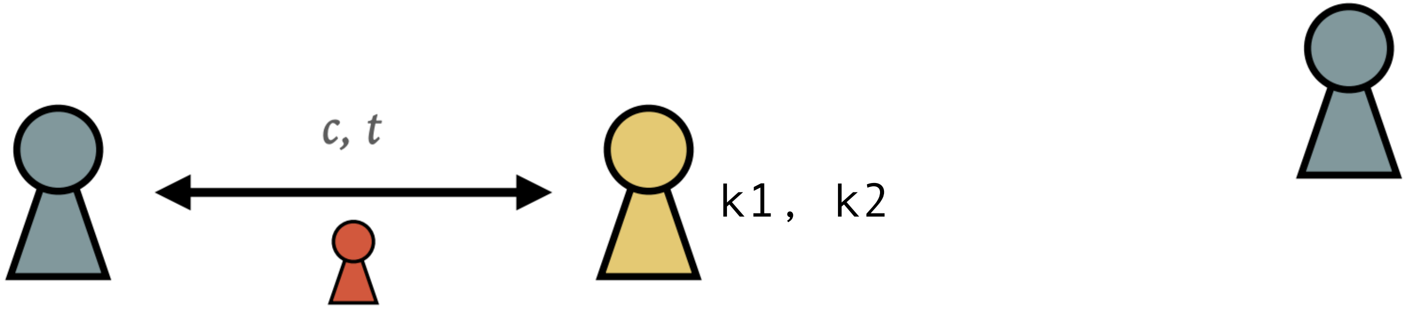
`Verify(k2, m, t) = 1`

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`

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

`k1, k2`

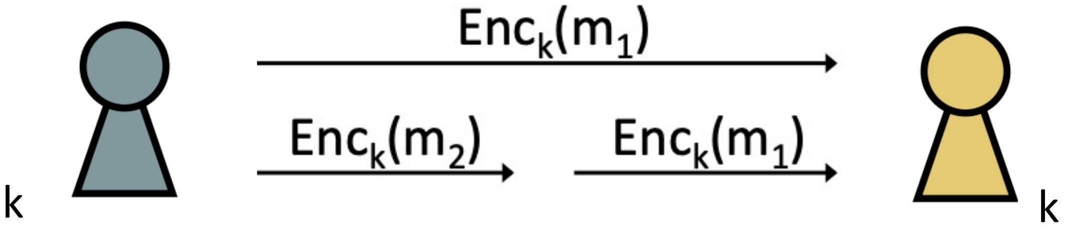
`Verify(k2, c, t) = 1`
`Dec(k1, c) = m`

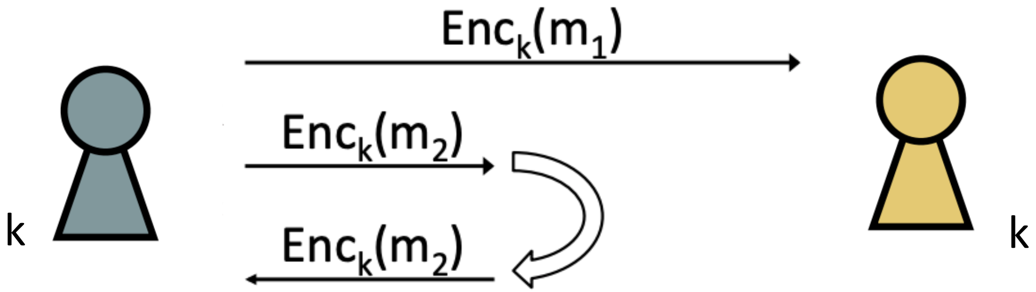
Encrypt then Authenticate: Is it secure?

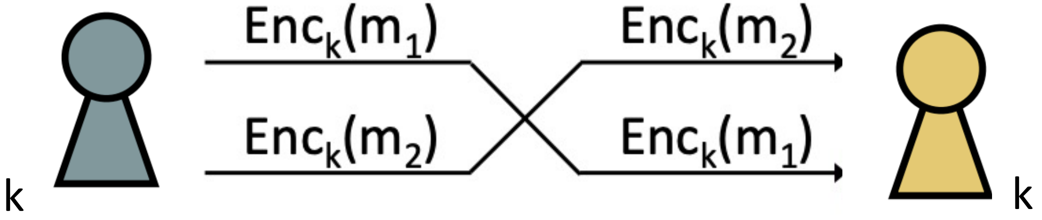
A. Yes, encryption is randomized with proper K, IV
B. No the tag might leak information
C. No the MAC is deterministic

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.



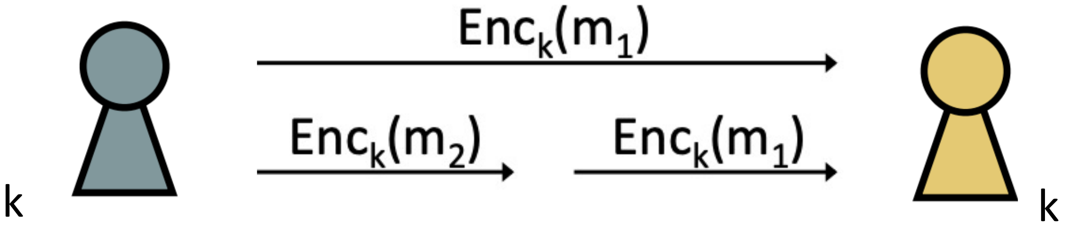




A. Yes
B. No

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?

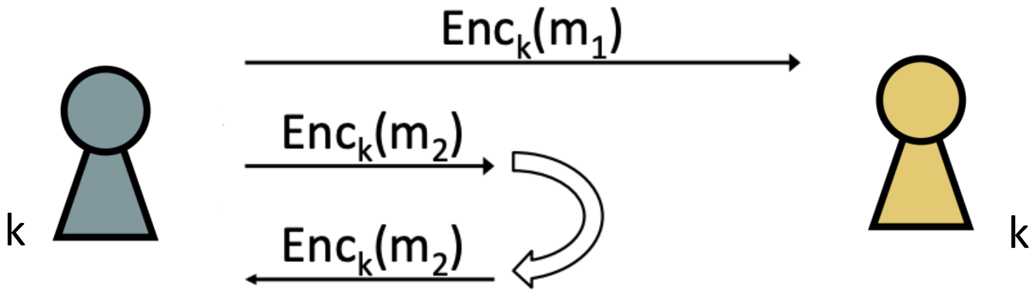
- Securely = secrecy and integrity
- Session = period of time over which parties are willing to maintain state.



$\text{Enc}_k(m_1)$

$\text{Enc}_k(m_2)$ $\text{Enc}_k(m_1)$

k k

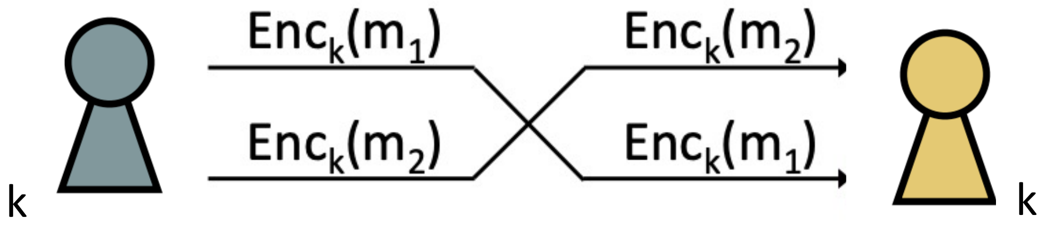


$\text{Enc}_k(m_1)$

$\text{Enc}_k(m_2)$

$\text{Enc}_k(m_2)$

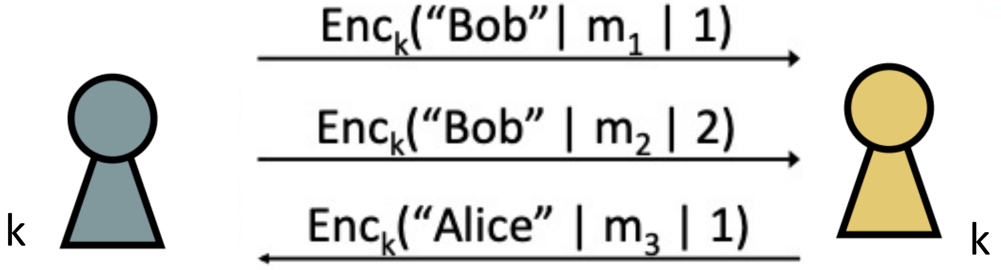
k k



$\text{Enc}_k(m_1)$ $\text{Enc}_k(m_2)$

$\text{Enc}_k(m_2)$ $\text{Enc}_k(m_1)$

k k



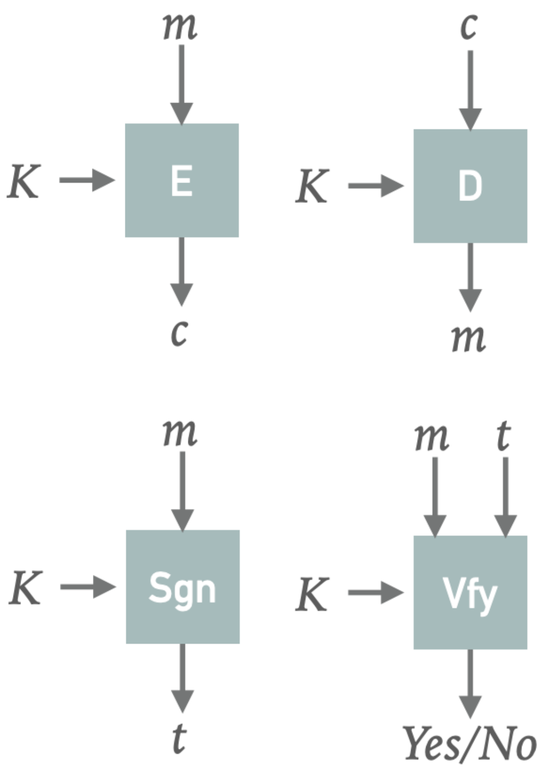
$\text{Enc}_k(\text{“Bob”} \mid m_1 \mid 1)$

$\text{Enc}_k(\text{“Bob”} \mid m_2 \mid 2)$

$\text{Enc}_k(\text{“Alice”} \mid m_3 \mid 1)$

k k

Symmetric Key Cryptography



CONFIDENTIALITY

Block ciphers

Deterministic ⟹ use IVs

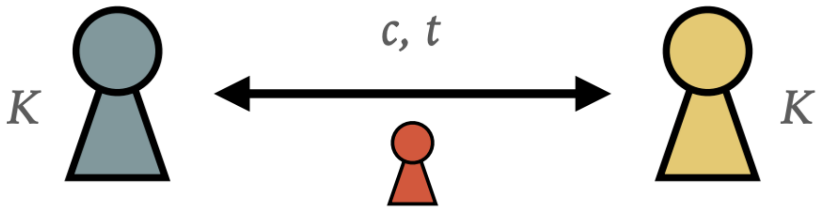
Fixed block size ⟹ use encryption "modes"

INTEGRITY

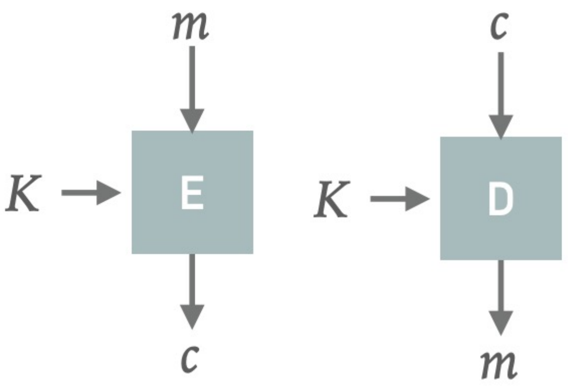
Message Authentication Codes (MACs)

Send (message, tag) pairs

Verify that they match



Symmetric Key Cryptography

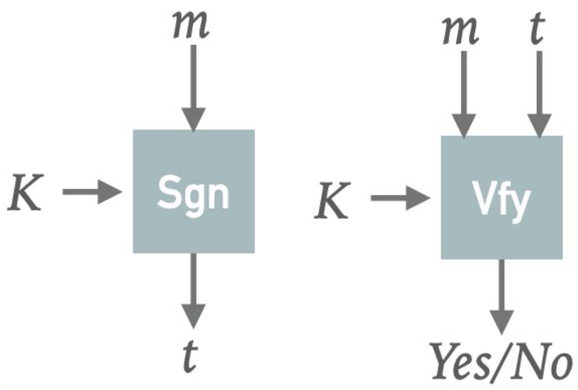


CONFIDENTIALITY

Block ciphers

Deterministic ⟹ use IVs

Fixed block size ⟹ use encryption "modes"



INTEGRITY

Message Authentication Codes (MACs)

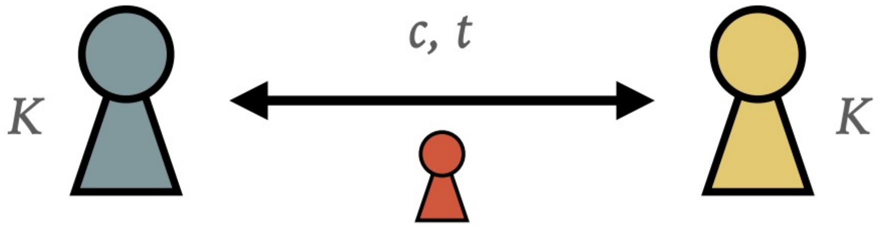
Send (message, tag) pairs

Verify that they match

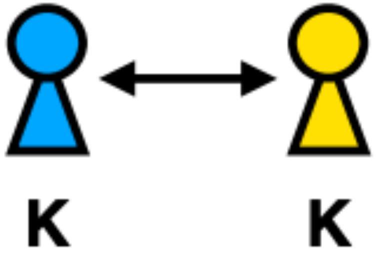
Next

How do we establish K?

How do we know with whom we are communicating?



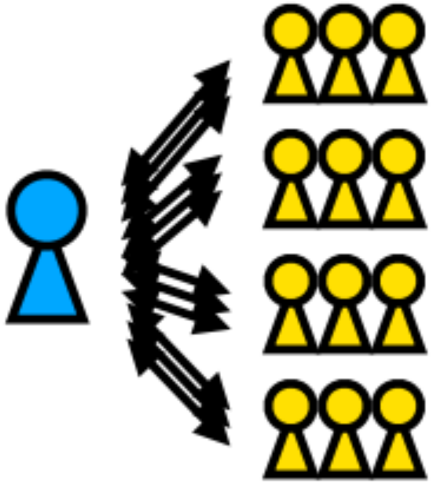
Shortcomings of symmetric key



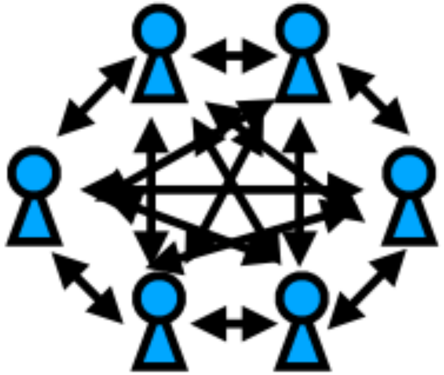
Establishing a pairwise key requires a **key exchange**, which requires both parties to be ***online***

Issue #1: Requires *pairwise* key exchanges

File downloads



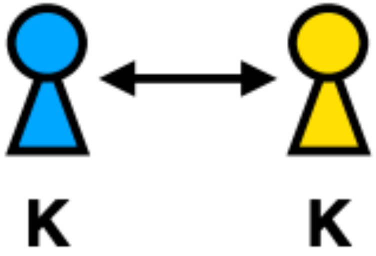
One-to-many:
O(N) key
exchanges



Email / chat

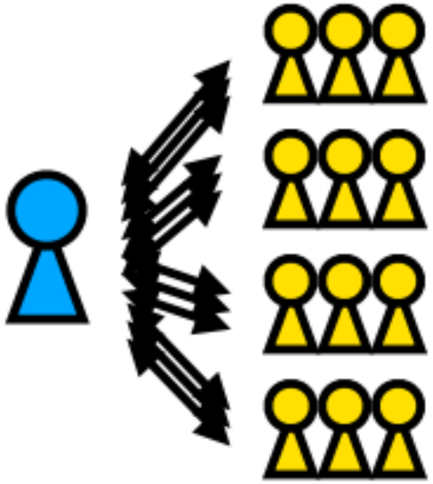
All-to-all:
$O(N^2)$ key
exchanges

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?

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
  - to an attacker, the shared key k established, is indistinguishable from a uniform key

Diffie-Helman Key Exchange

$x \bmod N$

g is a generator of mod N if

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

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

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

Given x and g, it is efficient to compute

$g^x \bmod N$

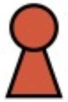
Given g and g^x, it is efficient to compute x

(simply take $\log_g g^x$)

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

Discrete log problem

Public knowledge: g and N



$g \quad N$

$g^a \bmod N$

$g^b \bmod N$

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

$g \quad N$

$g^a \bmod N$

$g^b \bmod N$

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

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:

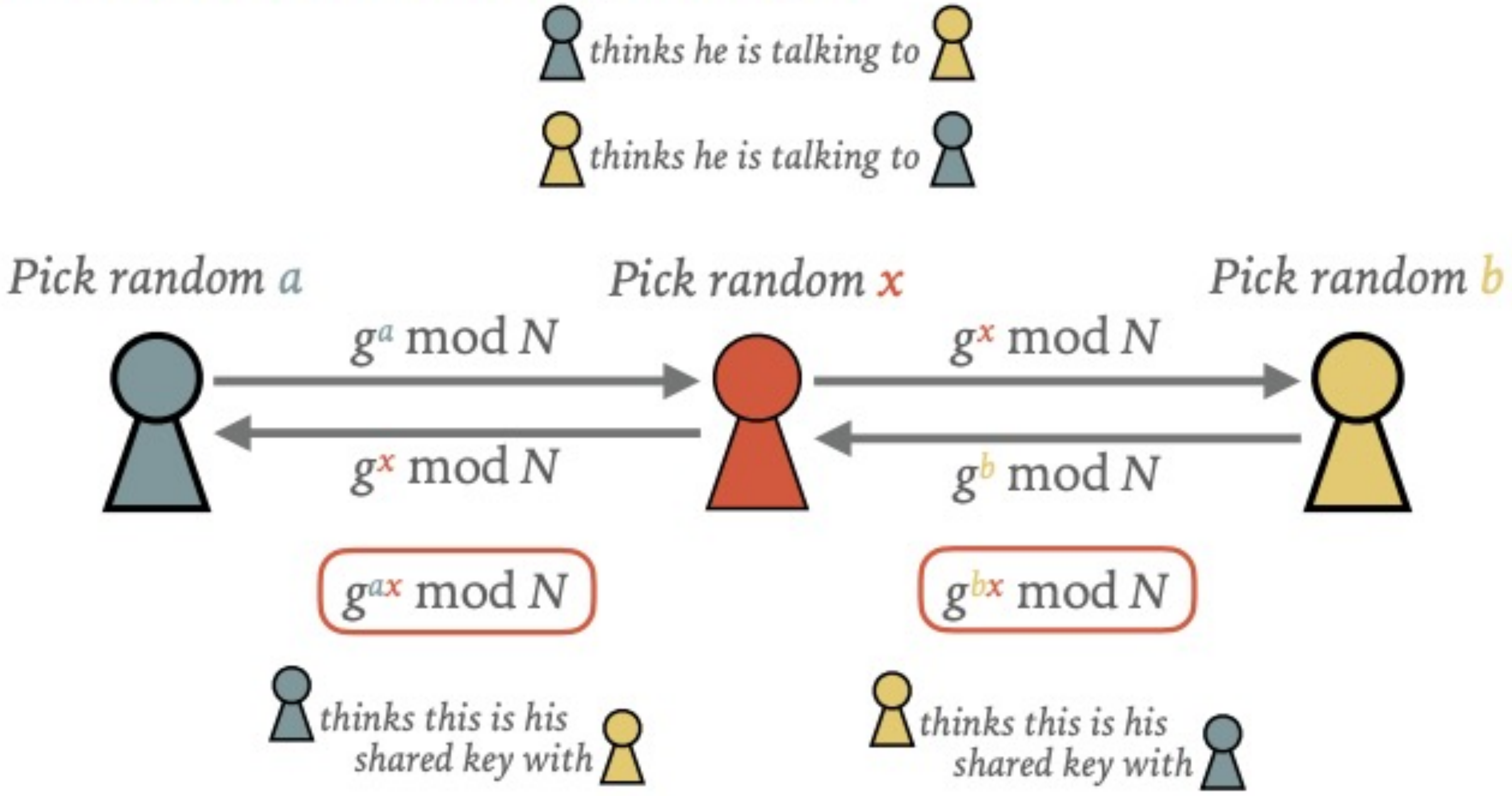
$$g^a \bmod N * g^b \bmod N \quad = g^{a+b} \bmod N$$

Key property:

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

But what about *active intermediaries*?

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

$g^a \bmod N$

$g^x \bmod N$

$g^x \bmod N$

$g^b \bmod N$

$g^{ax} \bmod N$

$g^{bx} \bmod N$

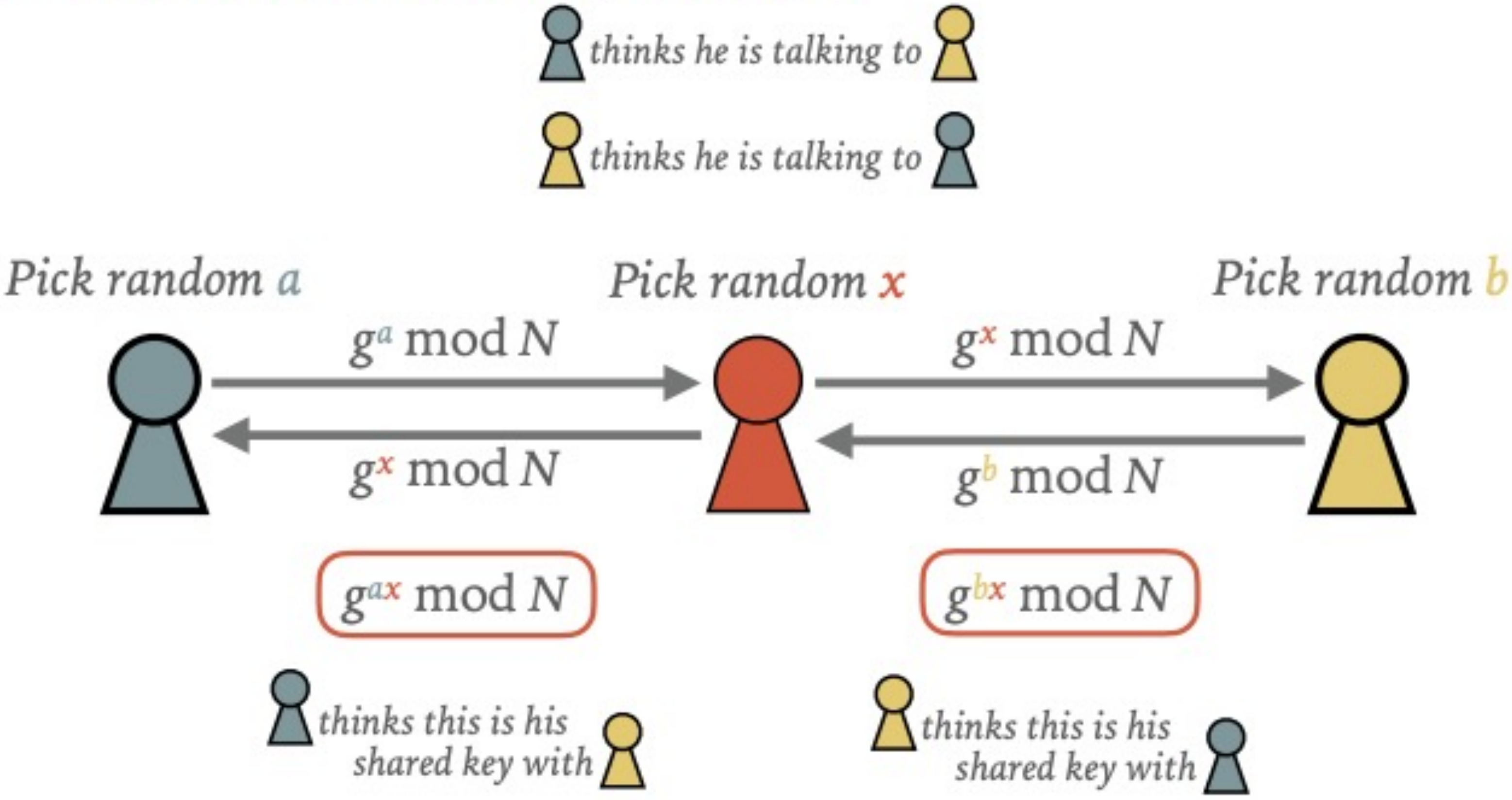
thinks this is his shared key with

thinks this is his shared key with

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.



thinks he is talking to

thinks he is talking to

Pick random a

Pick random x

Pick random b

$g^a \bmod N$

$g^x \bmod N$

$g^x \bmod N$

$g^b \bmod N$

$g^{ax} \bmod N$

$g^{bx} \bmod N$

thinks this is his shared key with

thinks this is his shared key with

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*

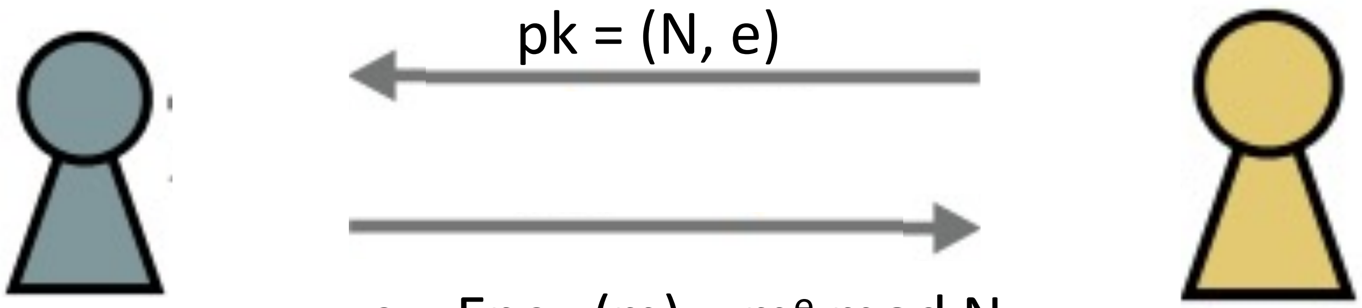
e *encryption exponent*

Secret key `sk`

p, q *primes*

d *decryption exponent*

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



$pk = (N, e)$

$c = Enc_{PK}(m) = m^e \bmod N$

$d = Dec_{SK}(c) = c^d \bmod N$

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.*

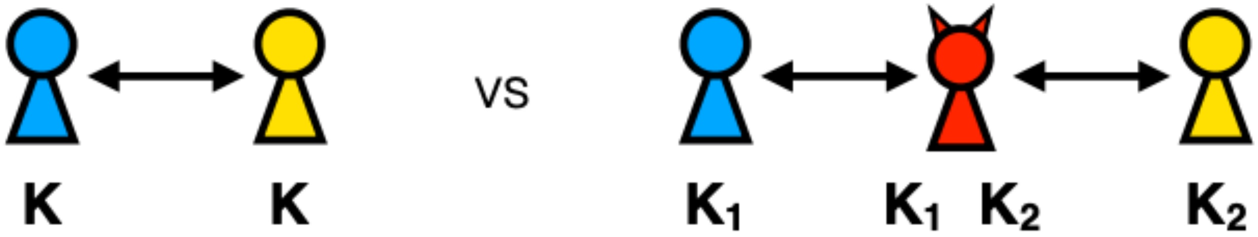
TO FIX THIS PROBLEM WE NEED…

BLACKBOX #5:
PUBLIC KEY CRYPTOGRAPHY

Shortcomings of symmetric key

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

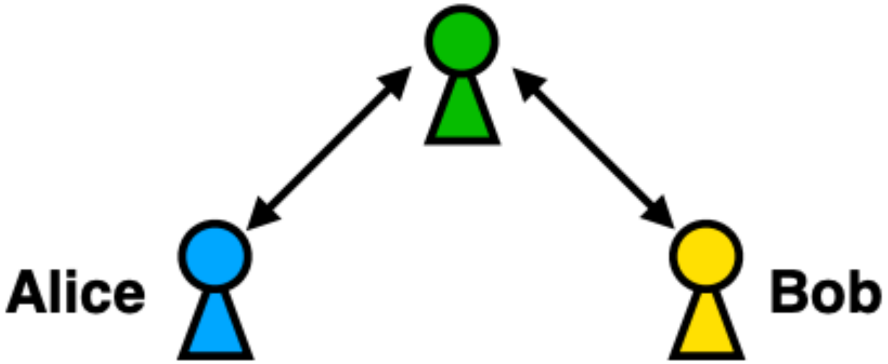
Diffie-Hellman is resilient to *eavesdropping*,
but ***not tampering***



Trusted Third Party

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

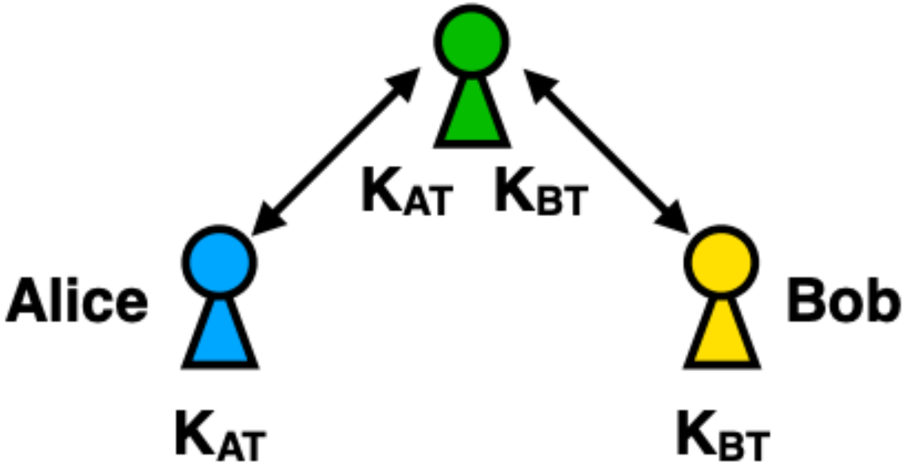
Trent: A *trusted* third party



Trusted Third Party

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

Trent: A *trusted* third party



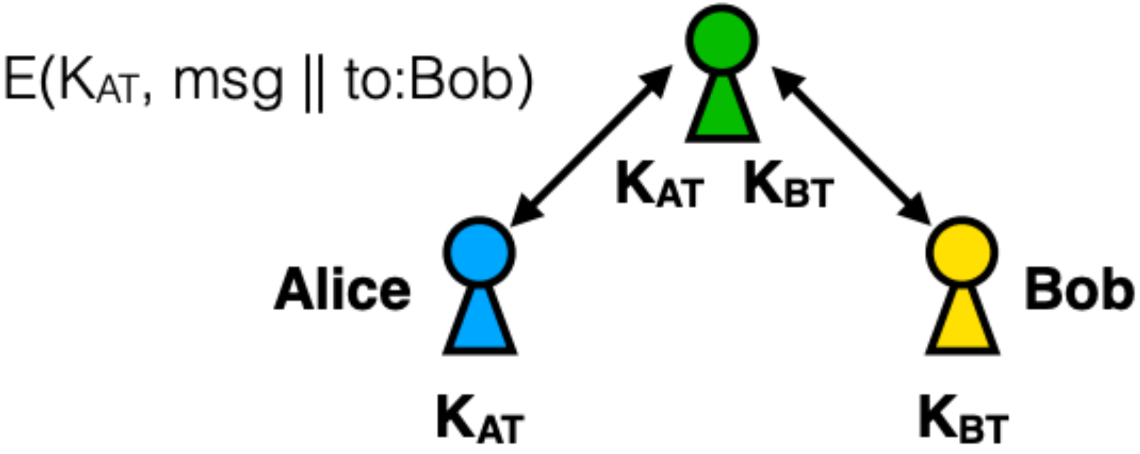
1. Everybody establishes a pairwise key with Trent

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*

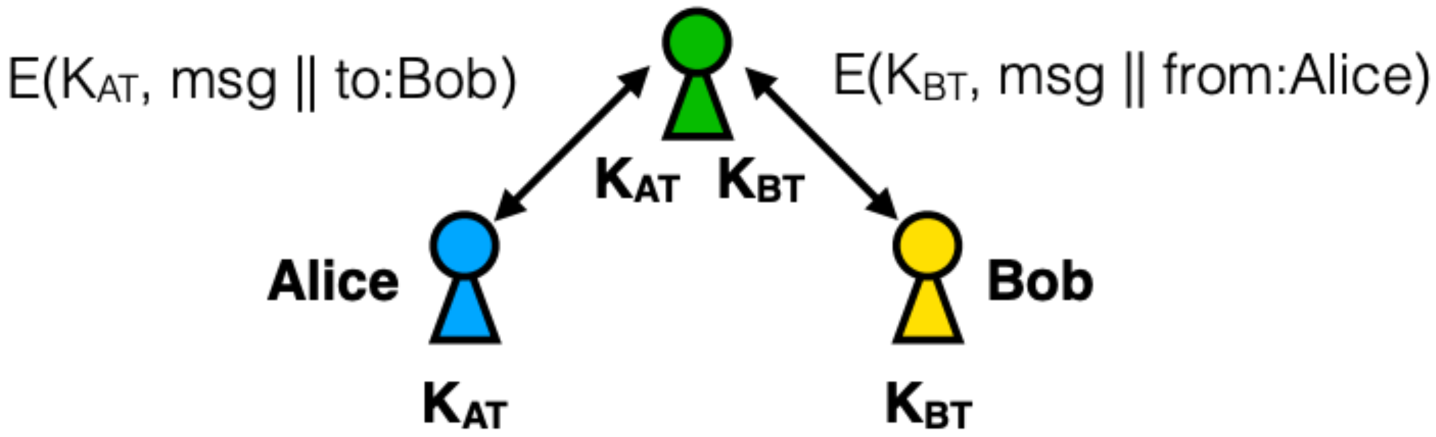
2. Trent validates each user's identity; includes in message

Good: *Authenticated communication*

Trusted Third Party

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

Trent: A *trusted* third party



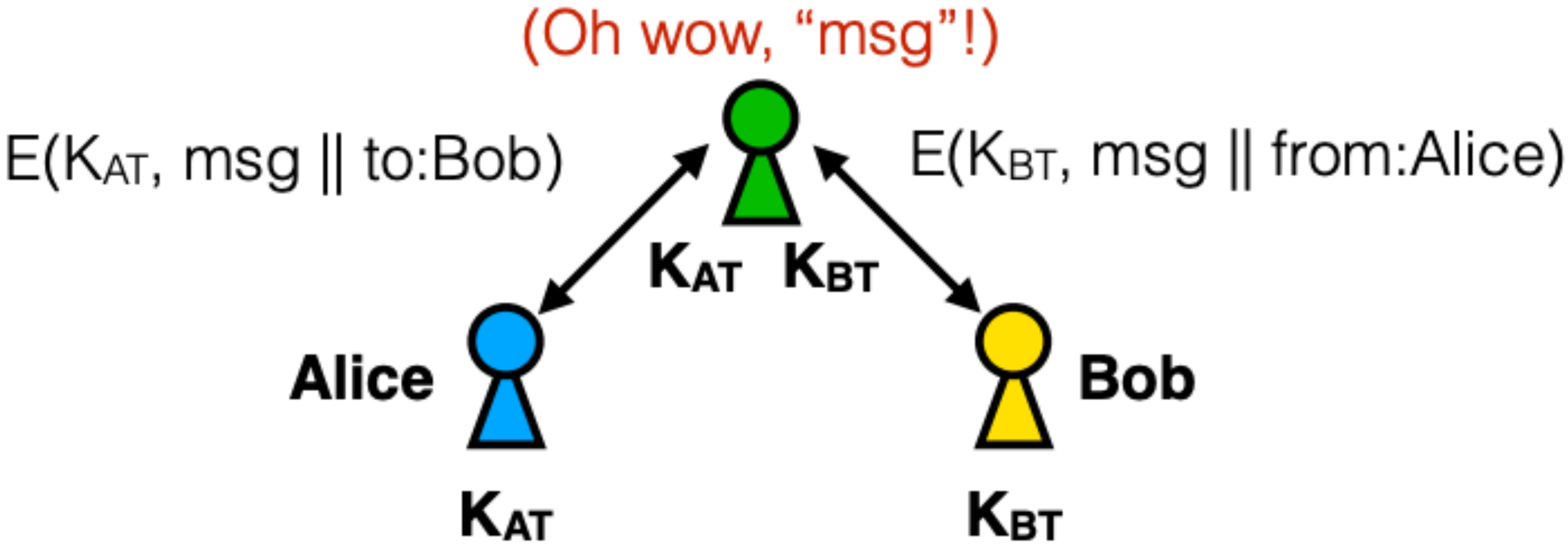
1. Everybody establishes a pairwise key with Trent
Good: *O(N) key exchanges*

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?

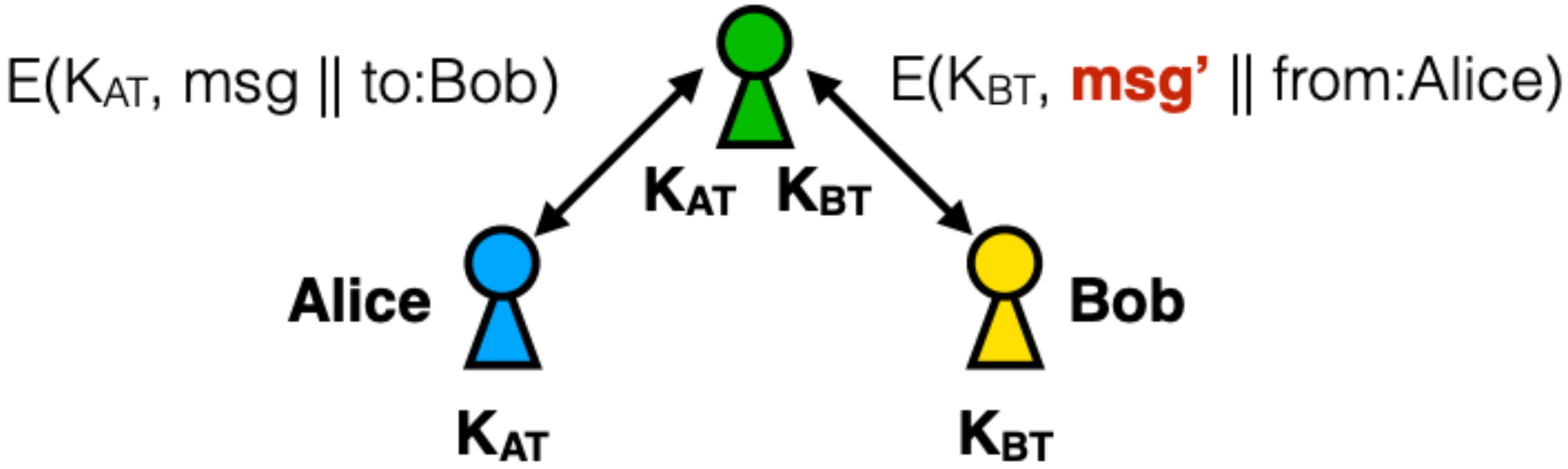
Just as “secure” meant nothing without an attack model, “trusted” means nothing without a **trust model**



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**



E(K_{AT}, msg || to:Bob)

E(K_{BT}, **msg’** || from:Alice)

K_{AT} **K_{BT}**

Alice

Bob

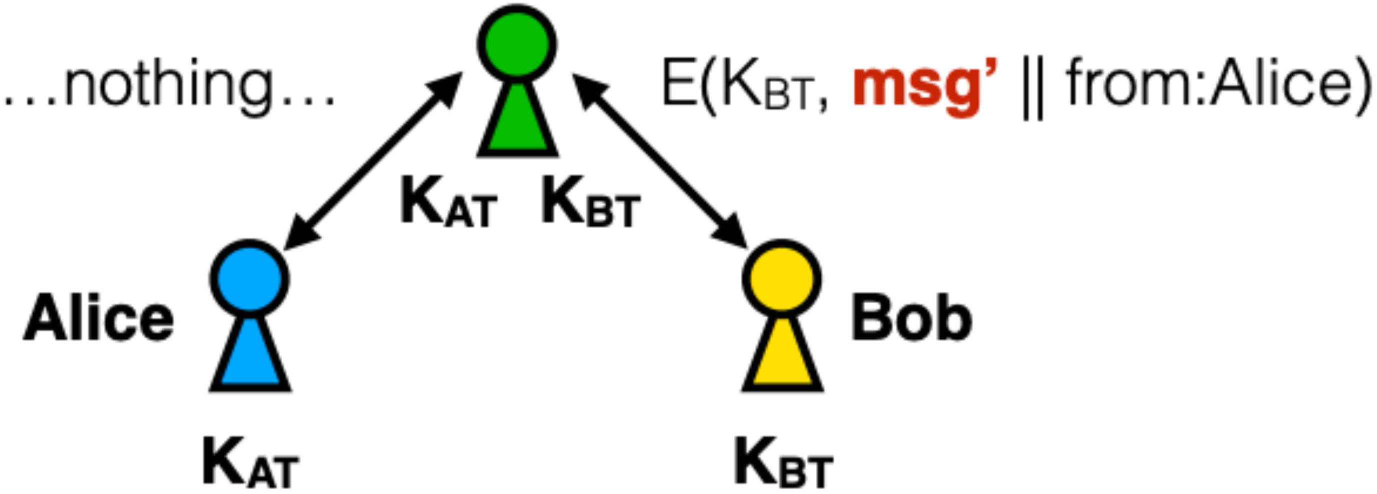
K_{AT}

K_{BT}

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

What are we trusting Trent not to do?

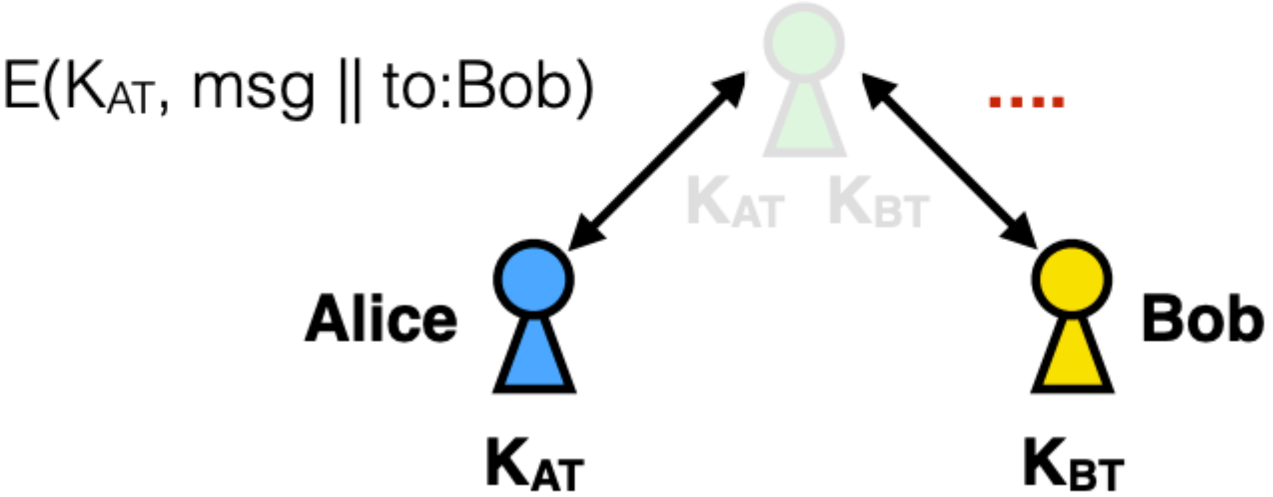
Just as "secure" meant nothing without an attack model, "trusted" means nothing without a **trust model**



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)

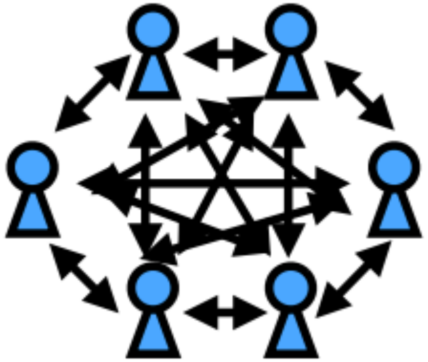
E() should approximate a one-way trapdoor function: cannot invert without access to SK

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)

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

Public key operations are *sloooow!*

Symmetric key operations are fast

Issues with public-key encryption

- No perfectly secret public-key encryption
- No deterministic public-key encryption scheme can be CPA-secure!
- CPA-security implies security for encrypting multiple messages as in the private-key case

Hybrid encryption

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

Obtain PK

Generate *symmetric* key K

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

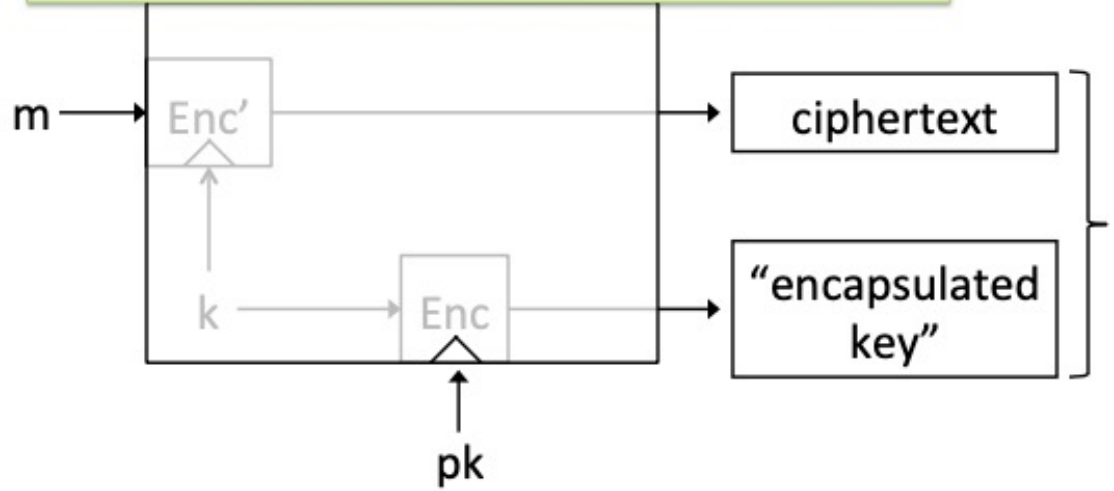
Public key Compute $c_K = E(PK, K)$

Send $c_K \| c_{msg}$

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

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

Decryption done in the obvious way



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

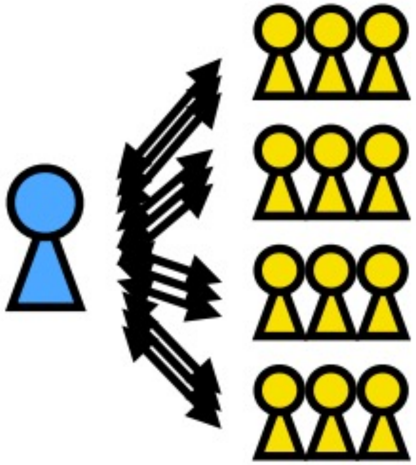
The speed and arbitrary message length of symmetric key

Protocols with public key cryptography

Goal: determine from whom a message came

Symmetric key

File downloads



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

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

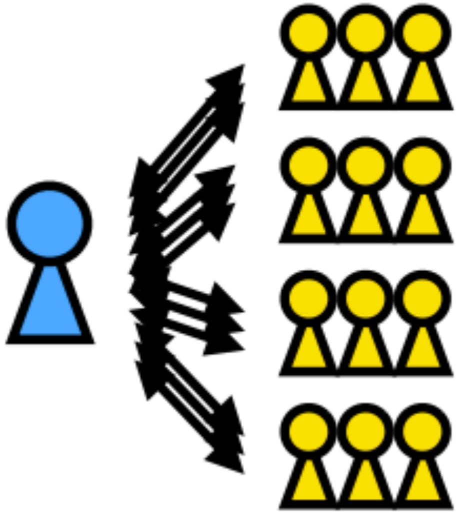
Same as with MACs: even after a chosen plaintext attack, the attacker cannot demonstrate an existential forgery

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, …)

Compute sig = Sgn(SK, msg)

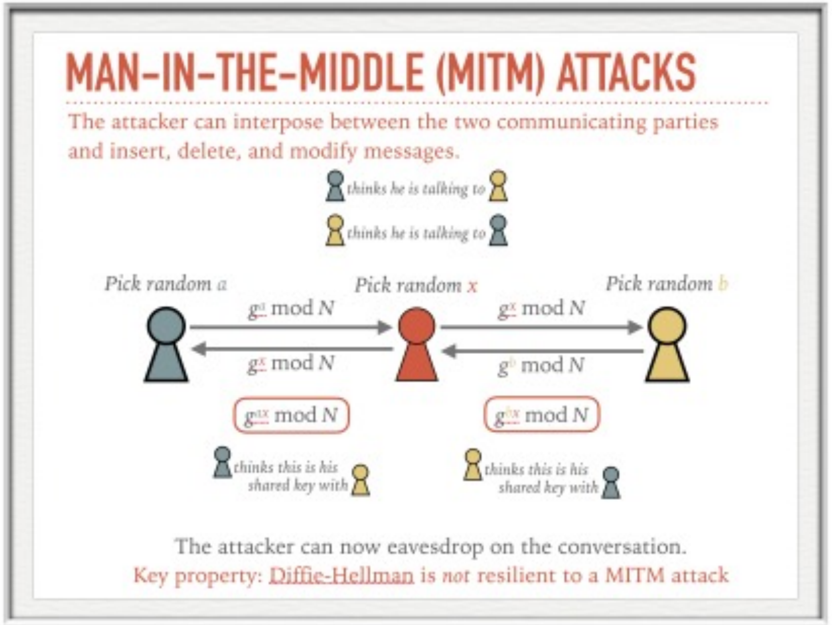
Publish msg || sig

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



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

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

Therefore, they were vulnerable to MITM attacks.

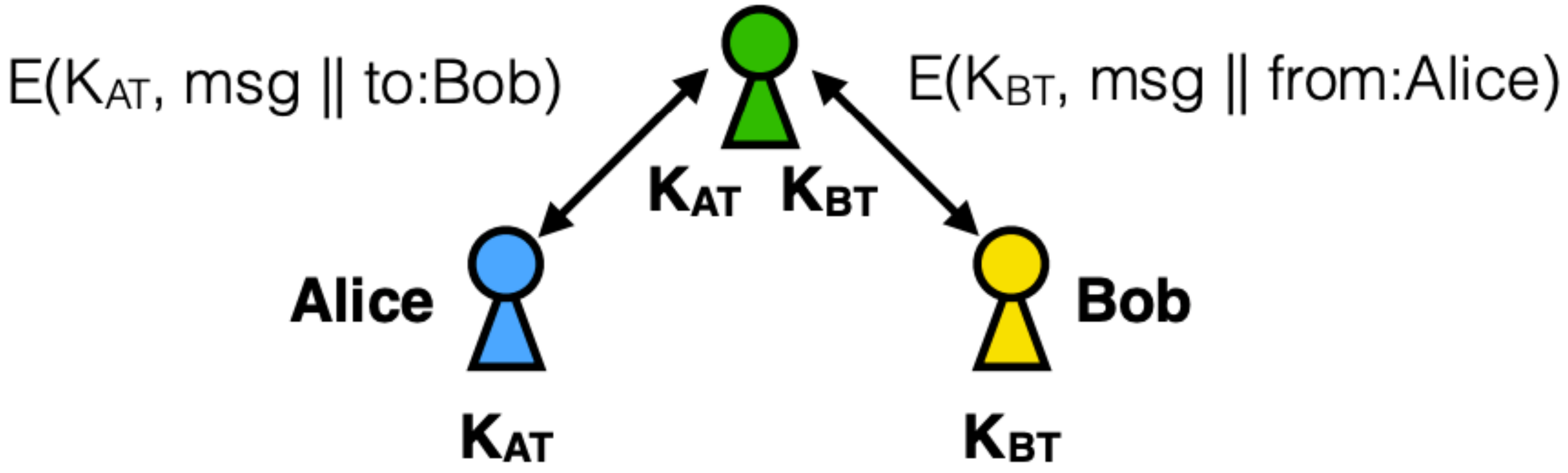
Certificates allow us to verify with whom we are communicating.

We will solve this by incorporating public key cryptography

Back to authentication

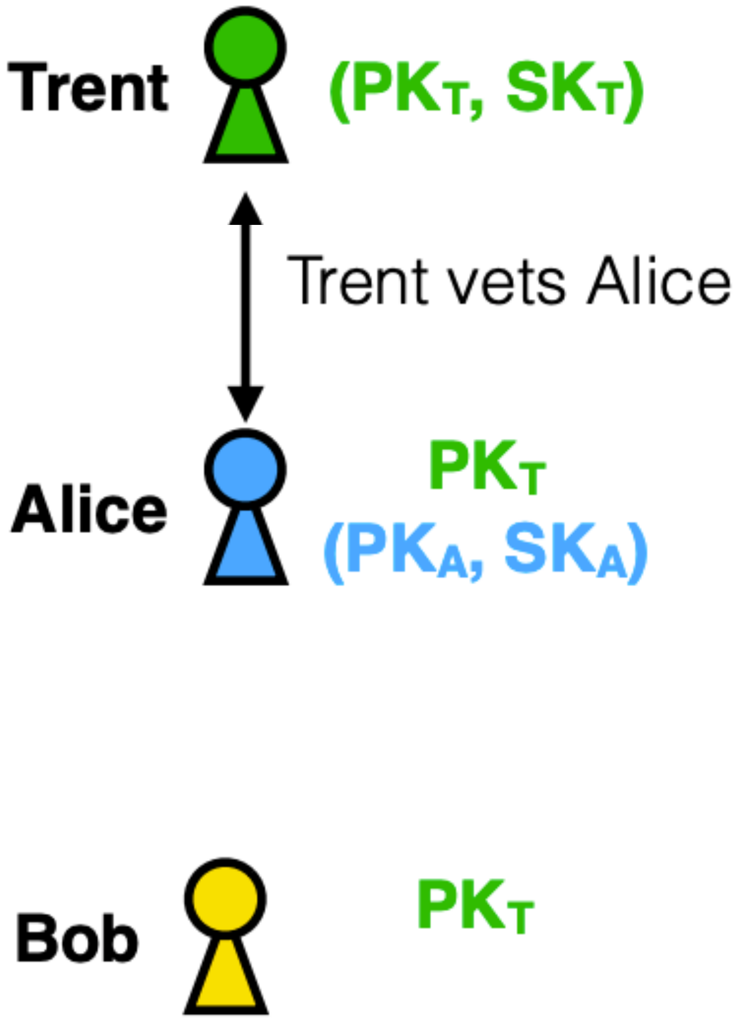
👤 Generate public/private key pair (PK,SK); publicize PK

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



Trent (PK_T, SK_T)

Trent vets Alice

Alice PK_T (PK_A, SK_A)

Bob PK_T

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

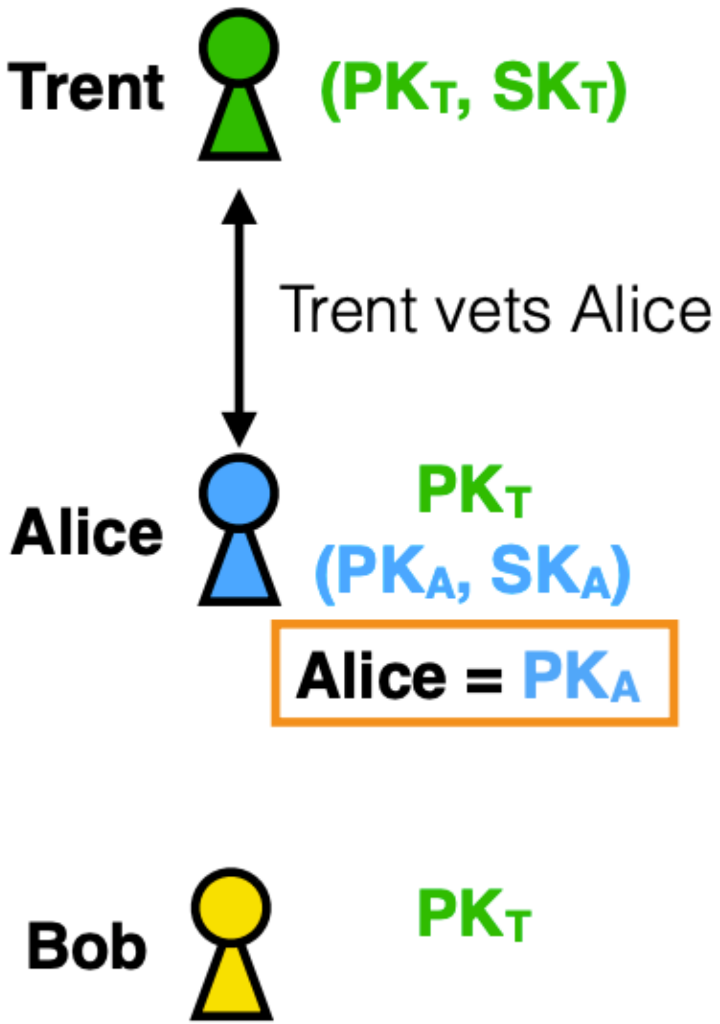
2. Alice generates a public/private key pair and asks Trent to bind her PK_A to her identity

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



Trent (PK_T, SK_T)

Trent vets Alice

Alice PK_T (PK_A, SK_A)

Alice = PK_A

Bob PK_T

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

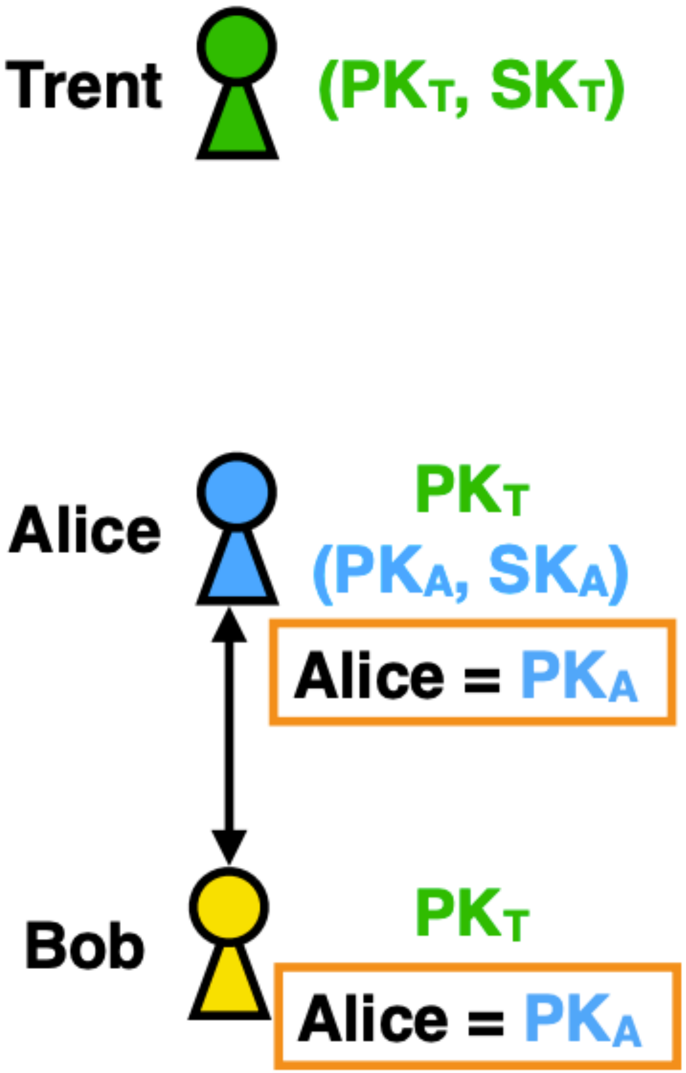
2. Alice generates a public/private key pair and asks Trent to bind her PK_A to her identity

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



Trent (PK_T, SK_T)

Alice PK_T (PK_A, SK_A)

Alice = PK_A

Bob PK_T

Alice = PK_A

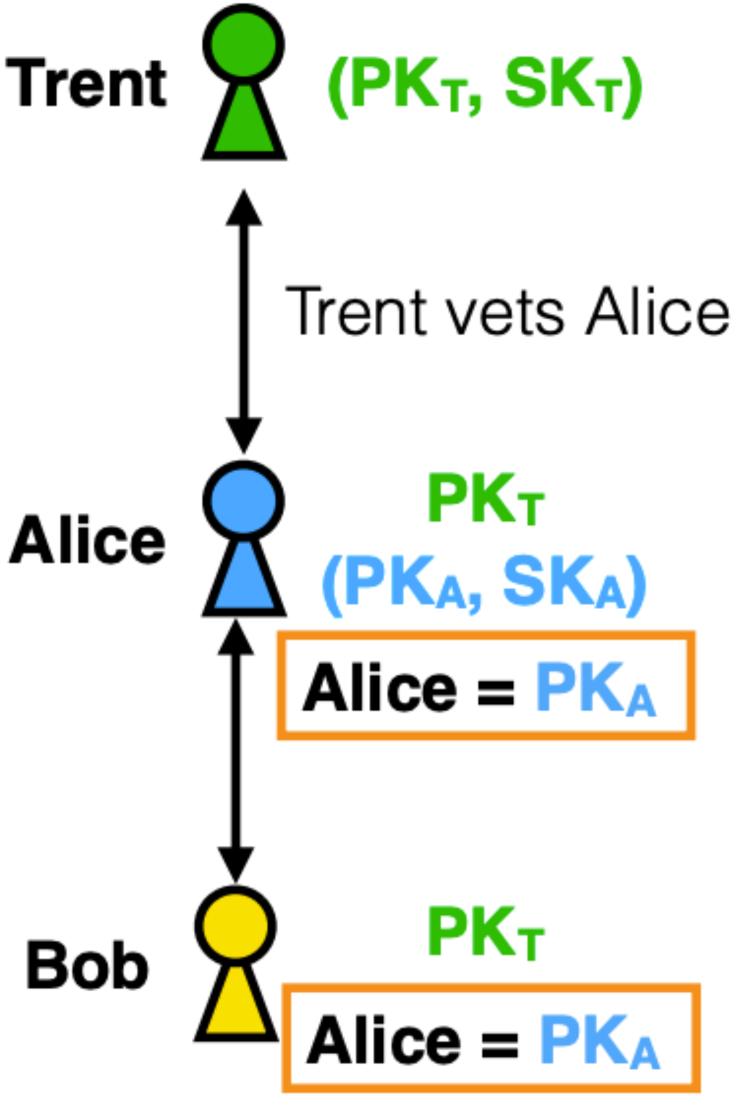
4. Alice makes her **certificate** publicly available (or Bob simply asks for it)

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



Trent (PK_T, SK_T)

Trent vets Alice

Alice PK_T

(PK_A, SK_A)

Alice = PK_A

Bob PK_T

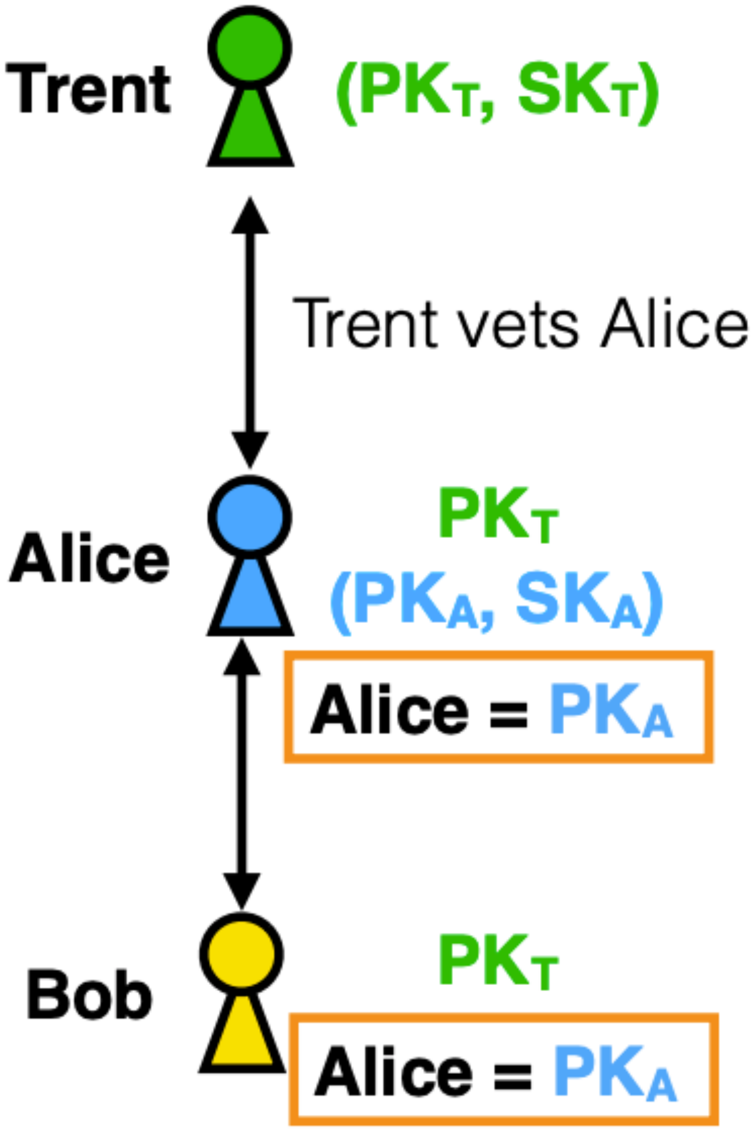
Alice = PK_A

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



Trent (PK_T, SK_T)

Trent vets Alice

Alice PK_T

(PK_A, SK_A)

Alice = PK_A

Bob PK_T

Alice = PK_A

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**):

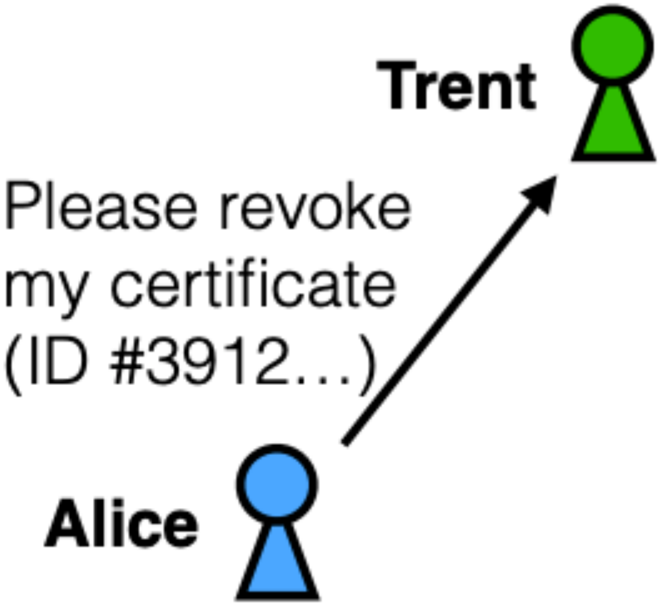
> “The owner of the secret key corresponding to **PK_A** is Alice”

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



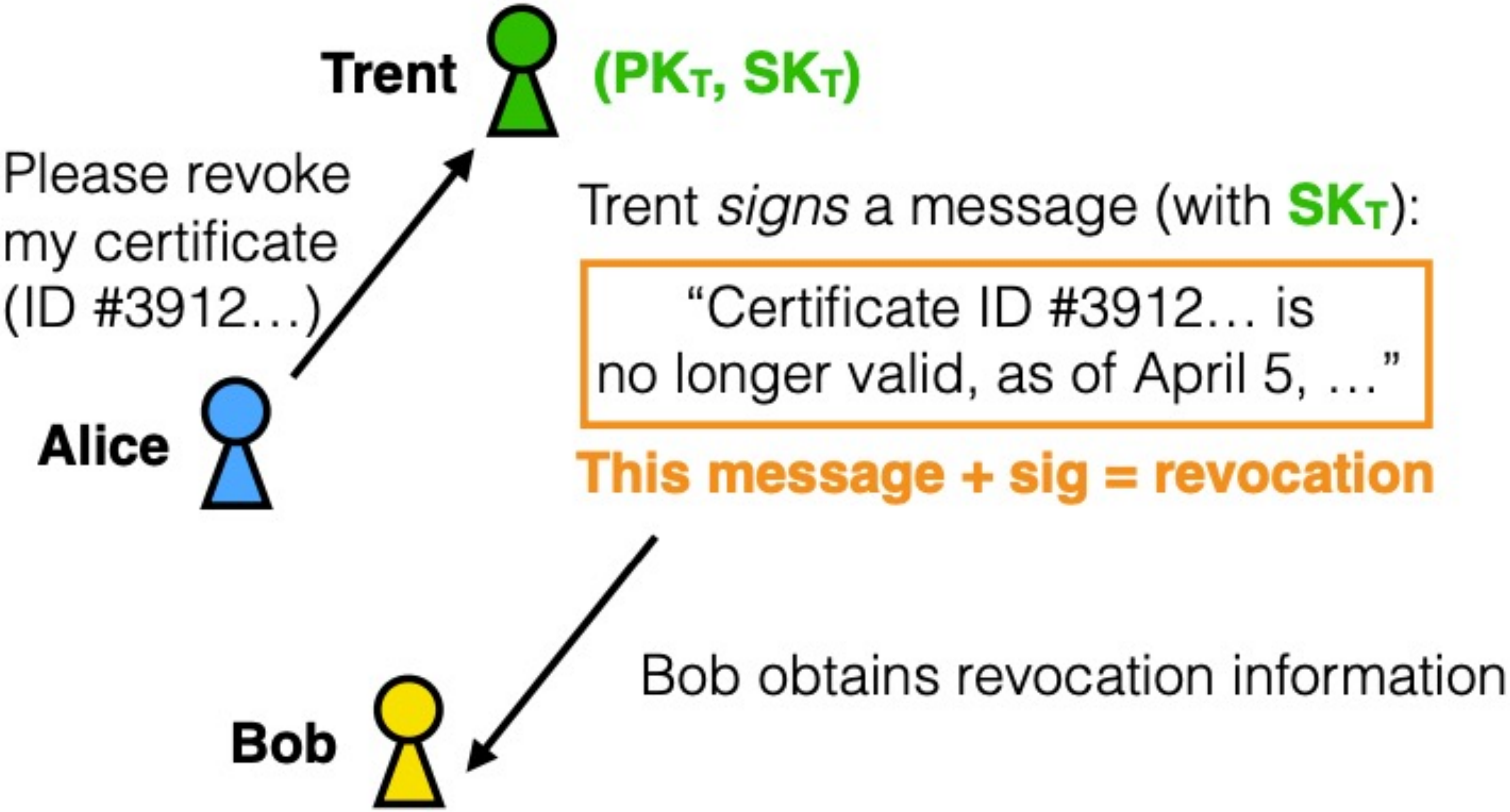
(PK_T, SK_T)

Please revoke my certificate (ID #3912…)

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

“Certificate ID #3912… is
no longer valid, as of April 5, …”

Certificate revocation



Trent **(PK_T, SK_T)**

Please revoke
my certificate
(ID #3912…)

Alice

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

“Certificate ID #3912… is
no longer valid, as of April 5, …”

This message + sig = revocation

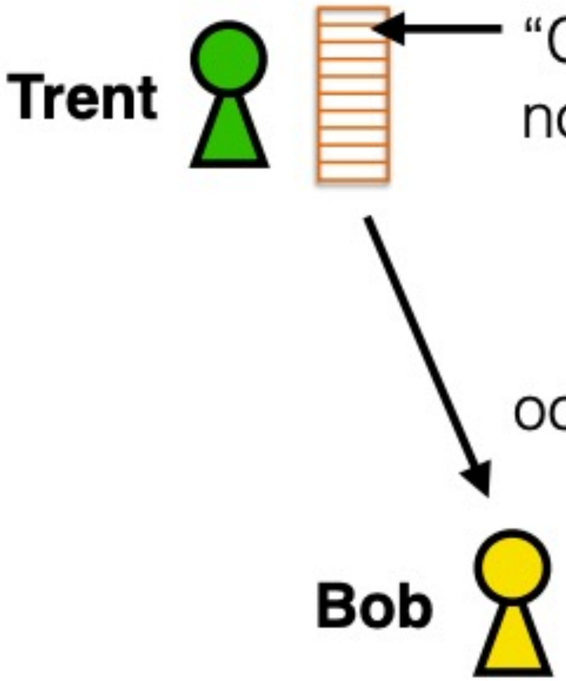
Bob obtains revocation information

Bob

Obtaining revocation data

Certificate Revocation Lists (CRLs)

A (often large) signed list of revocations



"Certificate ID #3912… is no longer valid, as of April 5, …"

Browsers and OSes occasionally download CRLs

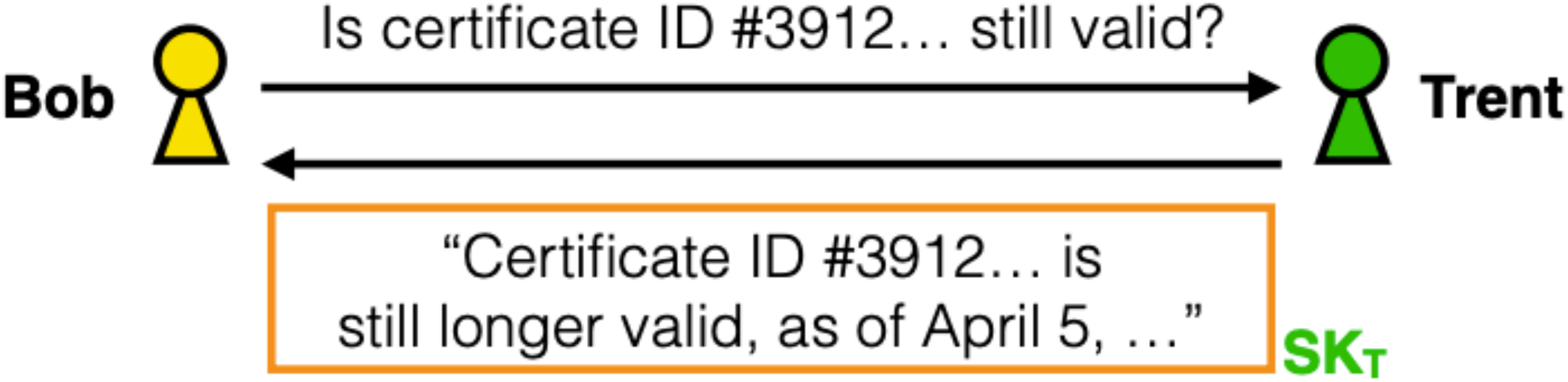
Disincentive: CRLs can be large, so it takes time & bandwidth

Result: delayed days/weeks/*forever*

Obtaining revocation data

Online Certificate Status Protocol (OCSP)

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



Is certificate ID #3912… still valid?

Bob

Trent

“Certificate ID #3912… is still longer valid, as of April 5, …”

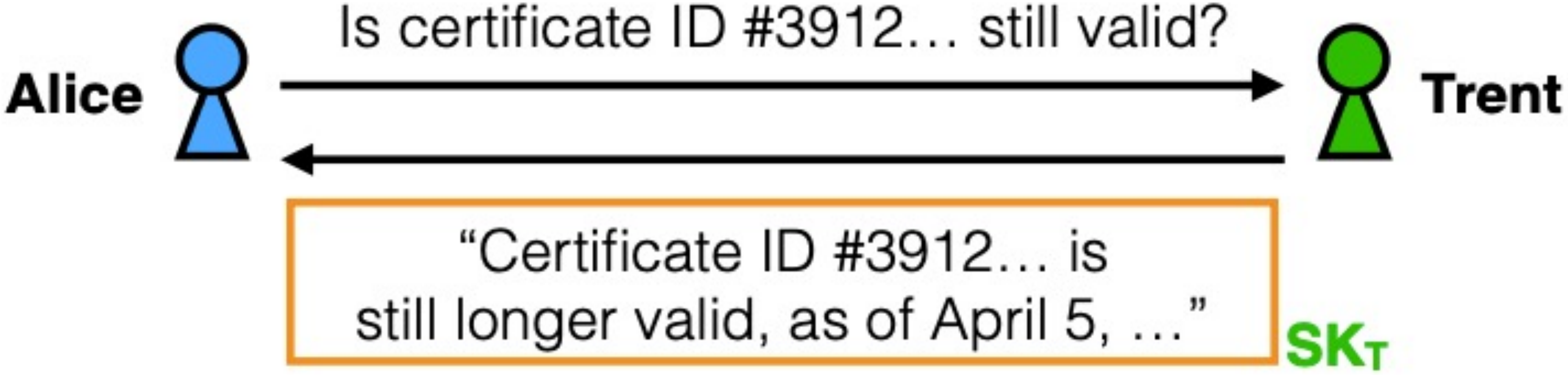
SK_T

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



Is certificate ID #3912… still valid?

"Certificate ID #3912… is
still longer valid, as of April 5, …" SK_T

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

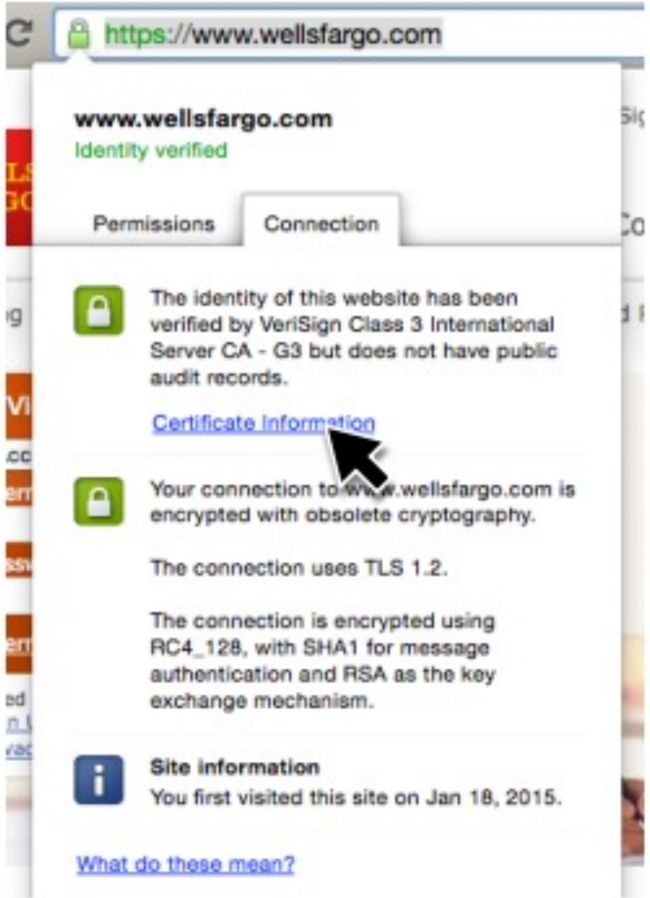
Bob's responsibility:
Check for revocations

Certificates in the wild

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

https://www.wellsfargo.com

Bank of America Corporation [US] https://www.bankofamerica.com



https://www.wellsfargo.com

www.wellsfargo.com

Identity verified

The identity of this website has been verified by VeriSign Class 3 International Server CA - G3 but does not have public audit records.

Certificate Information

Your connection to www.wellsfargo.com is encrypted with obsolete cryptography.

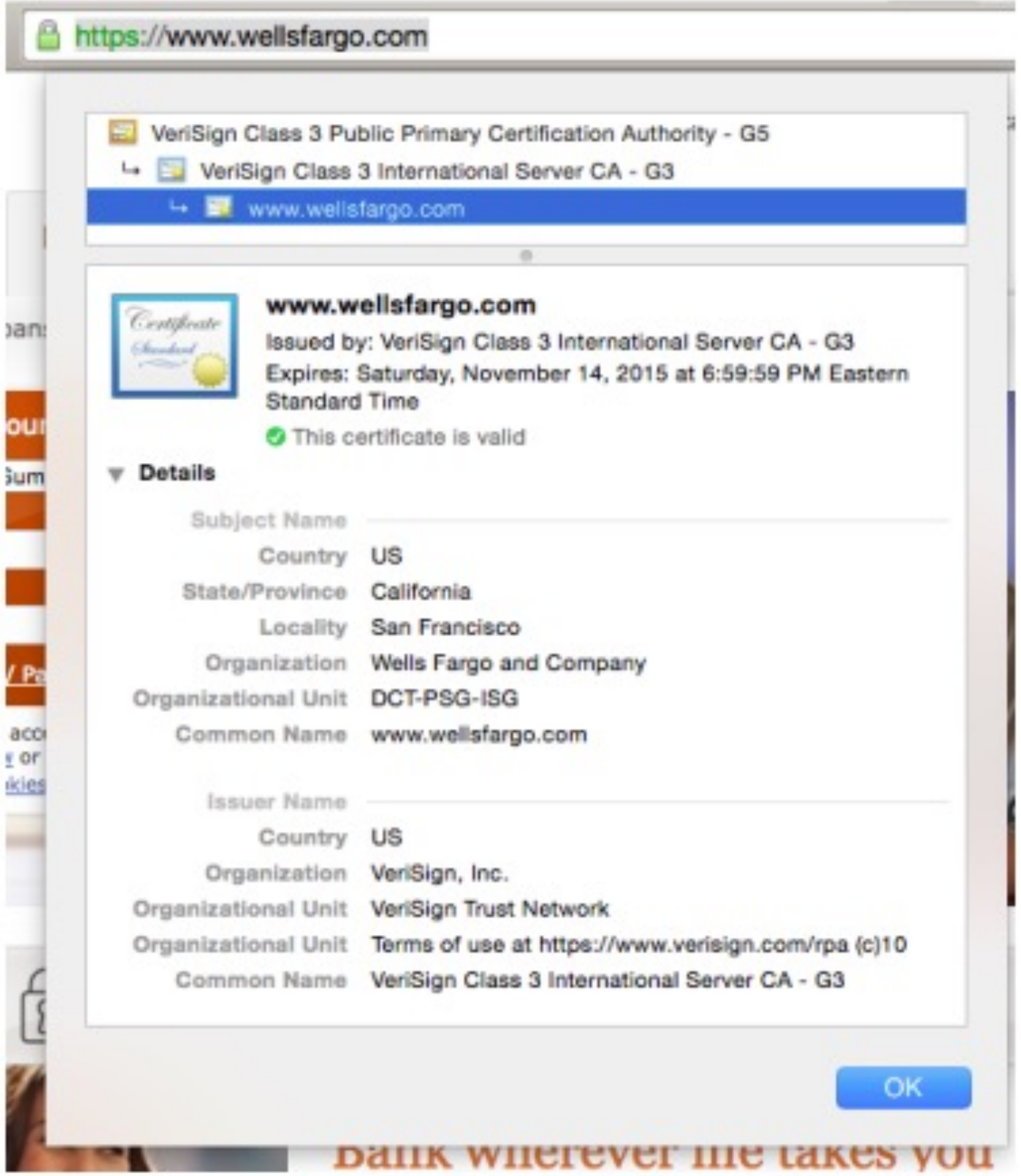
The connection uses TLS 1.2.

The connection is encrypted using RC4_128, with SHA1 for message authentication and RSA as the key exchange mechanism.

Site information

You first visited this site on Jan 18, 2015.

What do these mean?



https://www.wellsfargo.com

- VeriSign Class 3 Public Primary Certification Authority - G5
  - VeriSign Class 3 International Server CA - G3
    - www.wellsfargo.com

www.wellsfargo.com

Issued by: VeriSign Class 3 International Server CA - G3

Expires: Saturday, November 14, 2015 at 6:59:59 PM Eastern Standard Time

✓ This certificate is valid

▼ **Details**

Subject Name	
Country	US
State/Province	California
Locality	San Francisco
Organization	Wells Fargo and Company
Organizational Unit	DCT-PSG-ISG
Common Name	www.wellsfargo.com

Issuer Name	
Country	US
Organization	VeriSign, Inc.
Organizational Unit	VeriSign Trust Network
Organizational Unit	Terms of use at https://www.verisign.com/rpa (c)10
Common Name	VeriSign Class 3 International Server CA - G3

Certificate chain

Subject (who owns the public key)

Common name: the URL of the subject

Issuer (who verified the identity and signed this certificate)