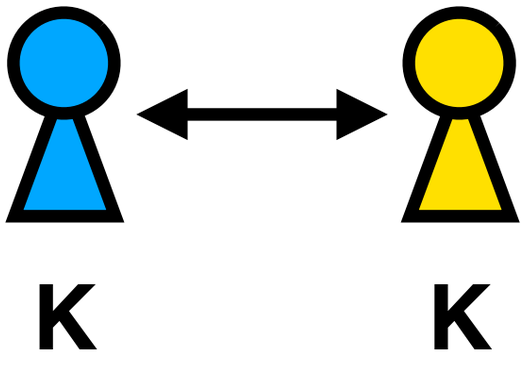
Public key cryptography and PKIs

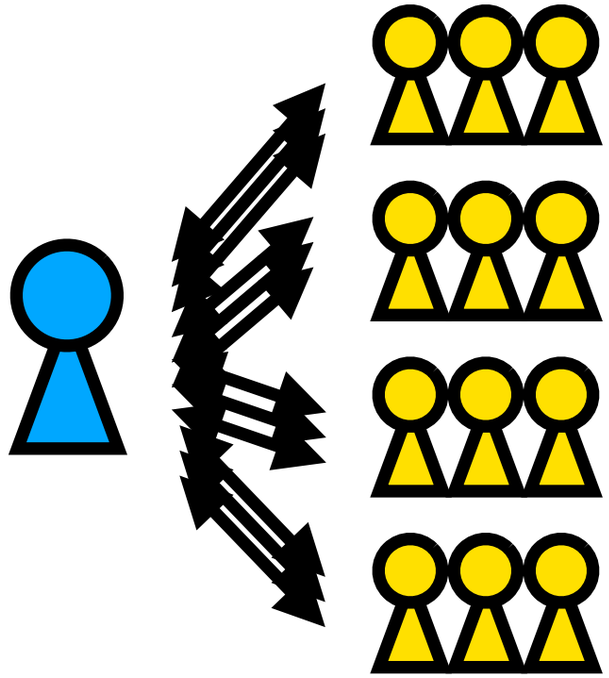
Shortcomings of symmetric key



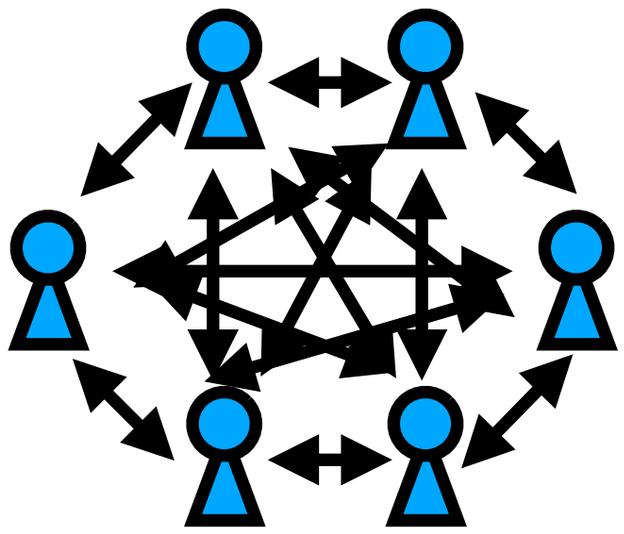
Establishing a pairwise key requires a **key exchange**, which requires both parties to be ***online***

Issue #1: Requires *pairwise* key exchanges

File downloads



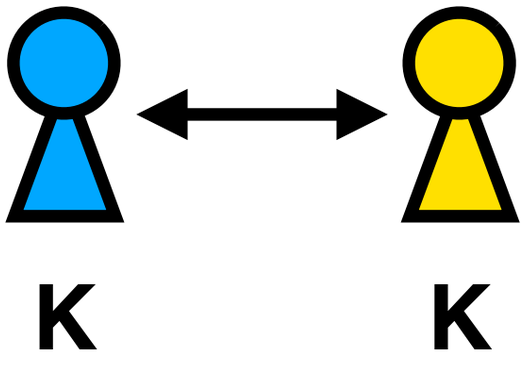
One-to-many:
O(N) key
exchanges



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

Email / chat

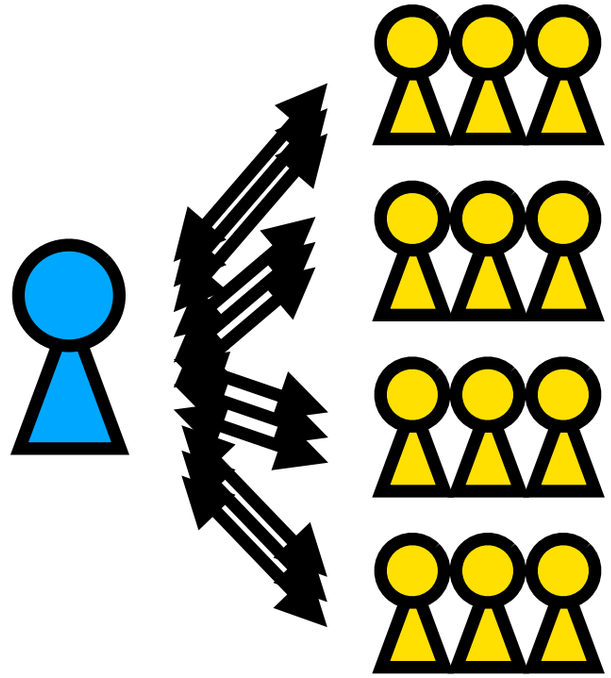
Shortcomings of symmetric key



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

Issue #2: Parties must be online

File downloads

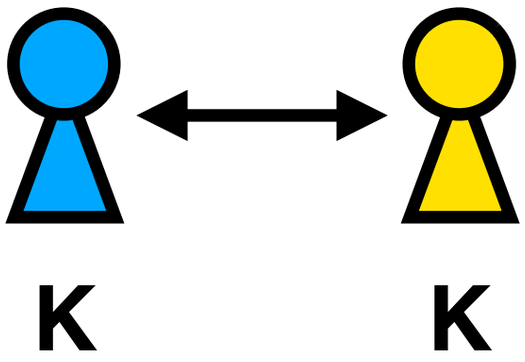


One-to-many: O(N) key exchanges

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?

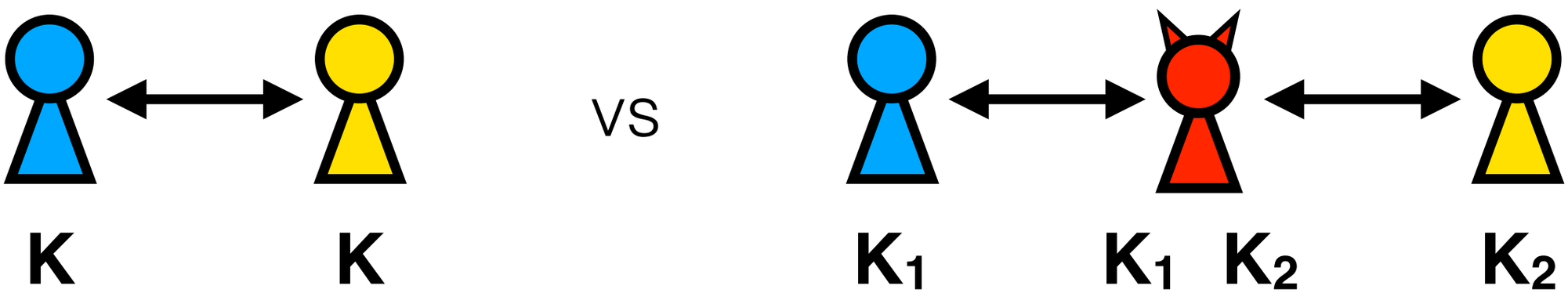
Shortcomings of symmetric key



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

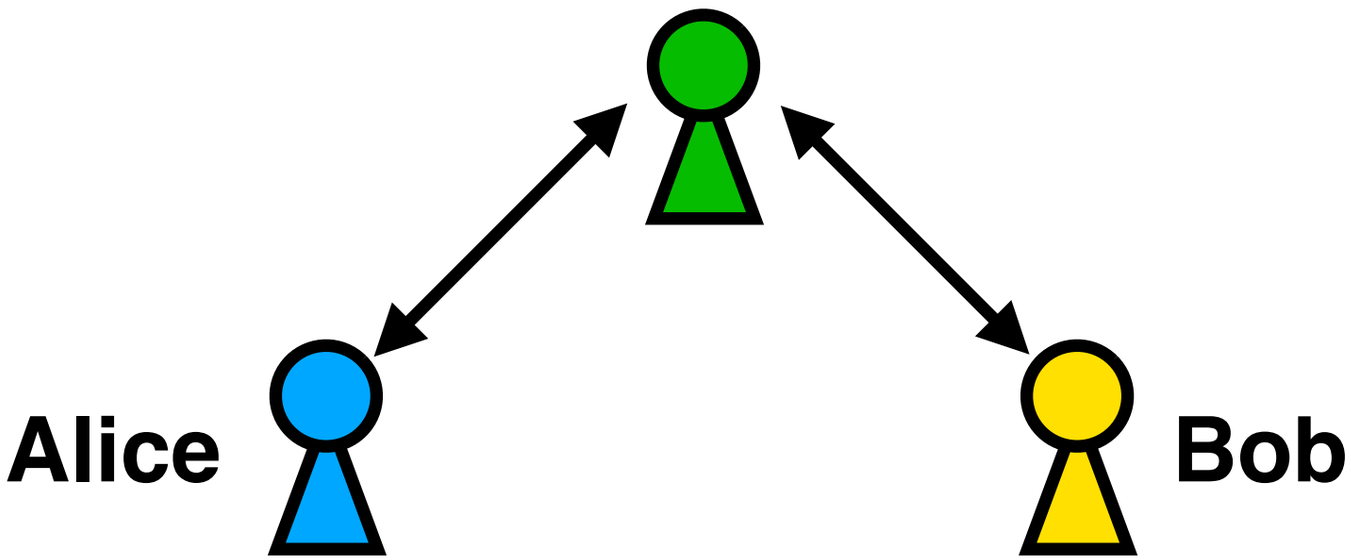
Issue #3: How do you know to whom you're talking?

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



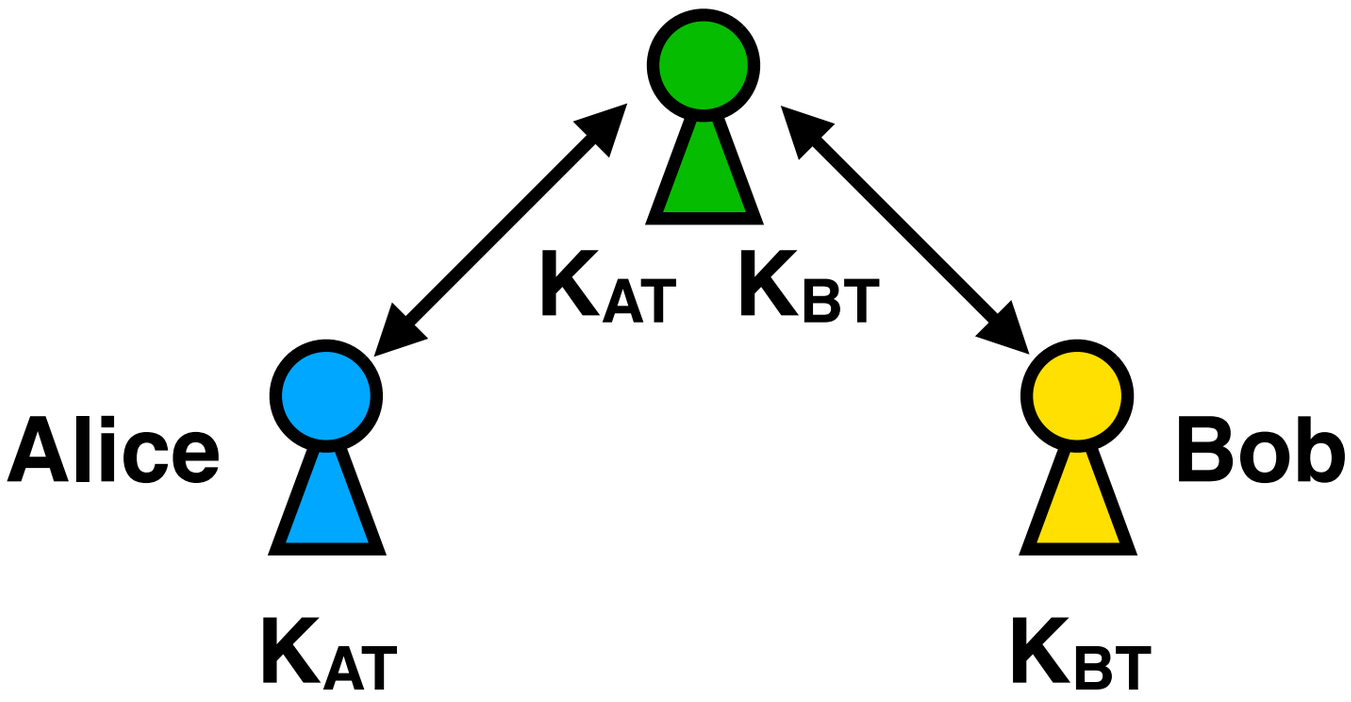
A protocol that solves this with ***trust***

Trent: A *trusted* third party



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

Trent: A *trusted* third party

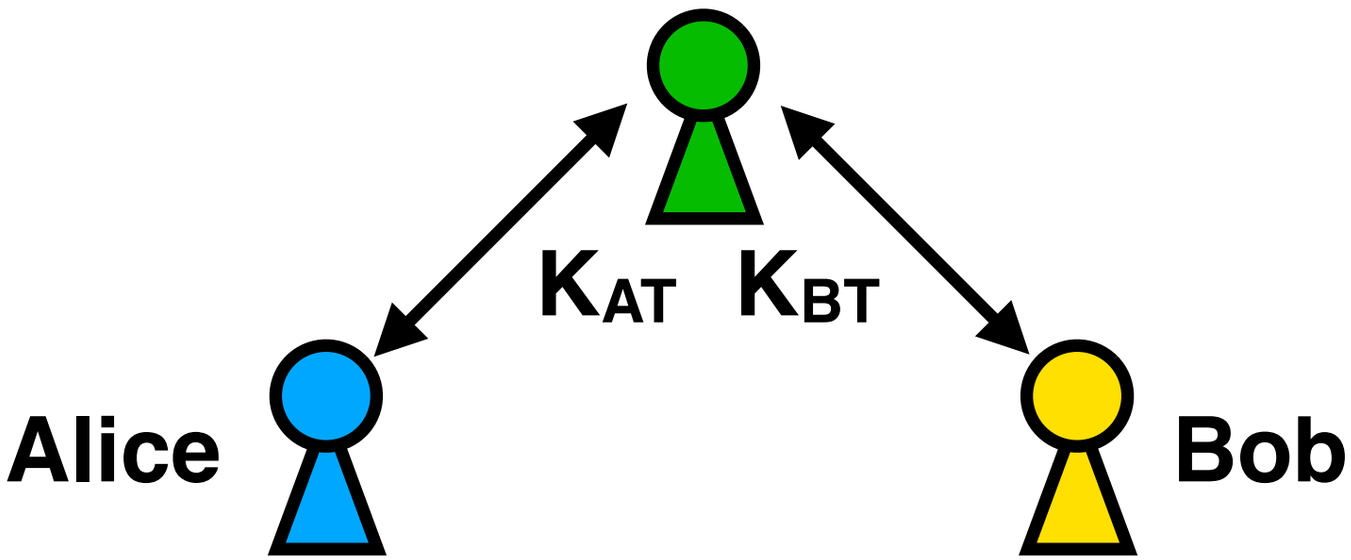


1. Everybody establishes a pairwise key with Trent

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

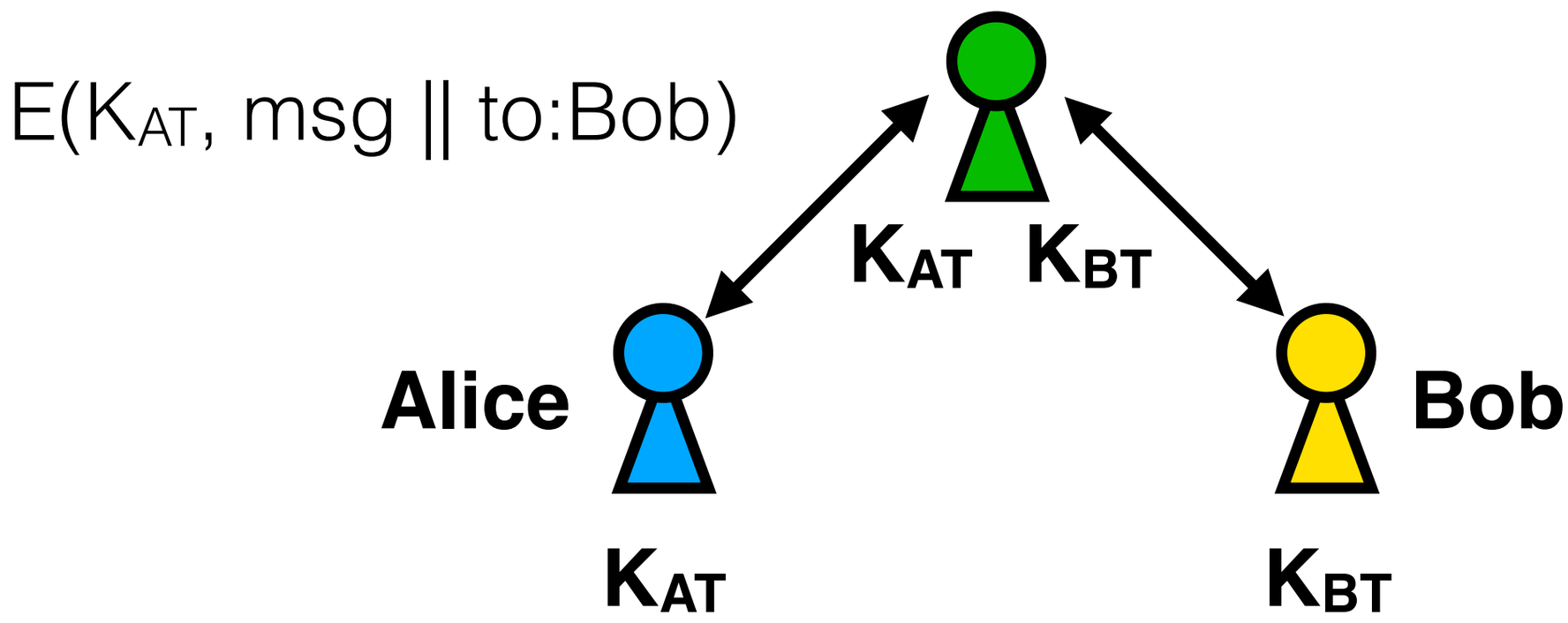
Good: *O(N) key exchanges*

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

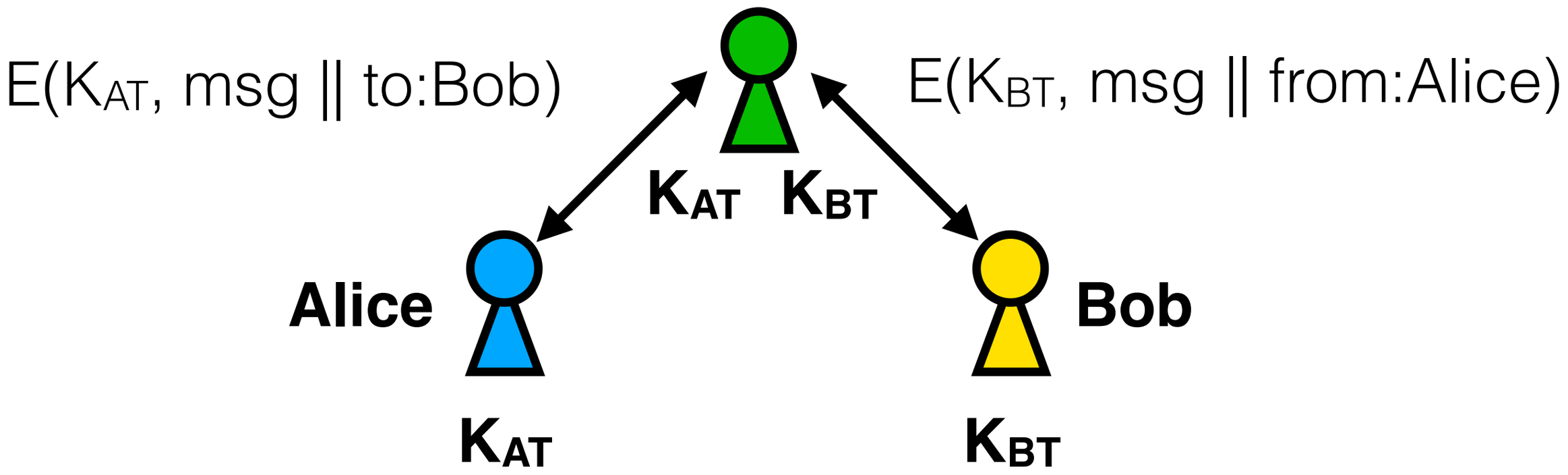
Good: *O(N) key exchanges*

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

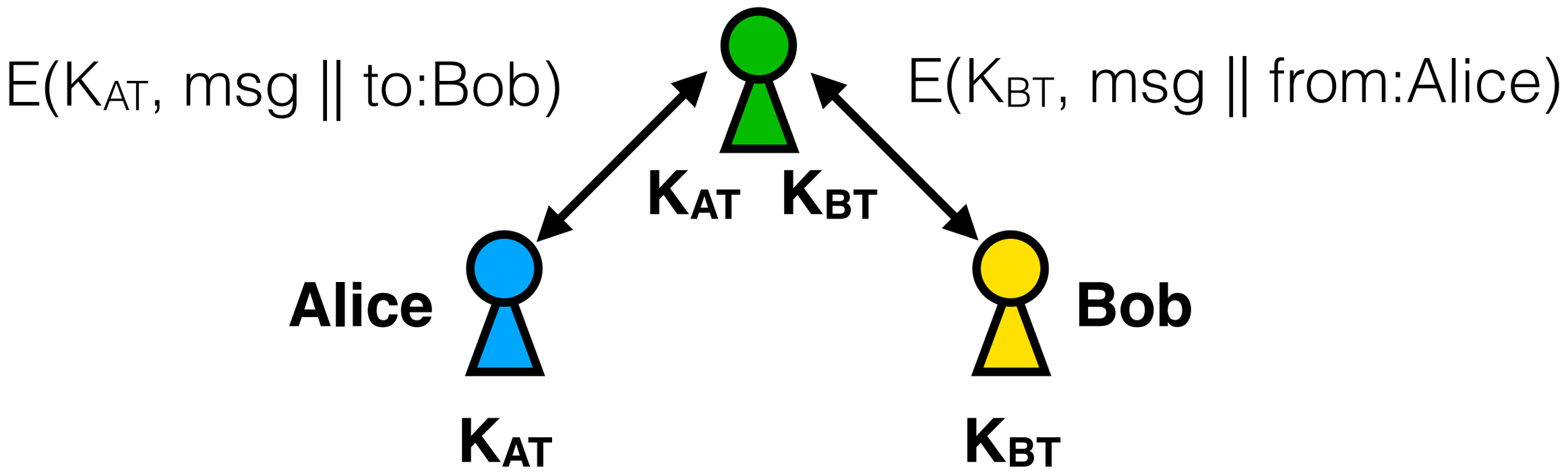
Good: *O(N) key exchanges*

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*

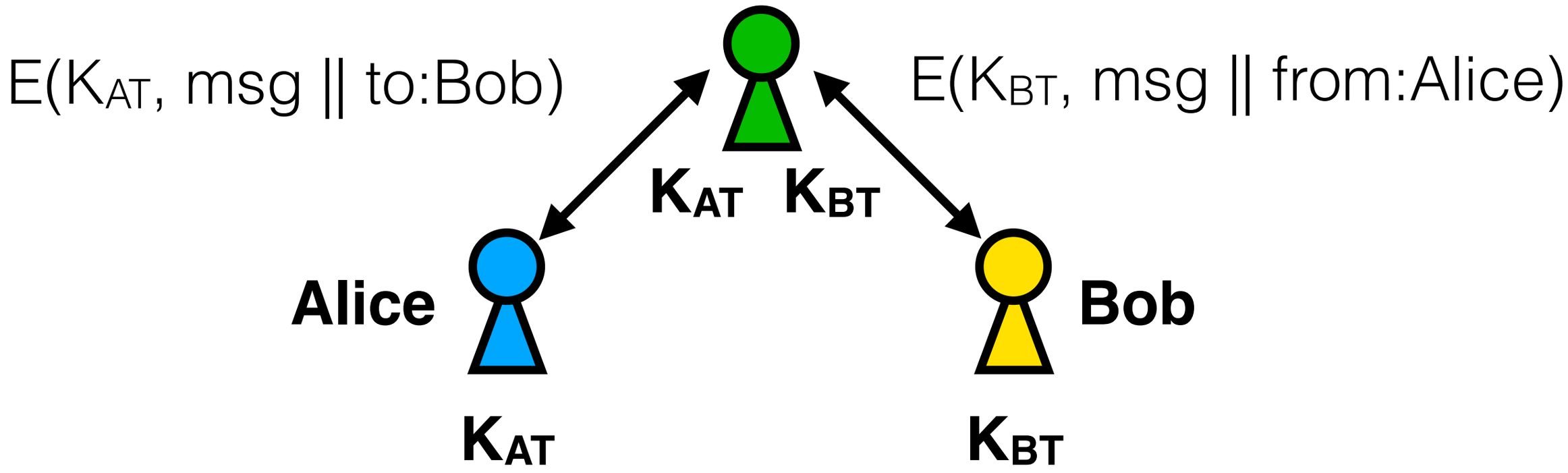
2. Trent validates each user's identity; includes in message

Good: *Authenticated communication*

Bad: All messages get sent through Trent

What are we trusting Trent not to do?

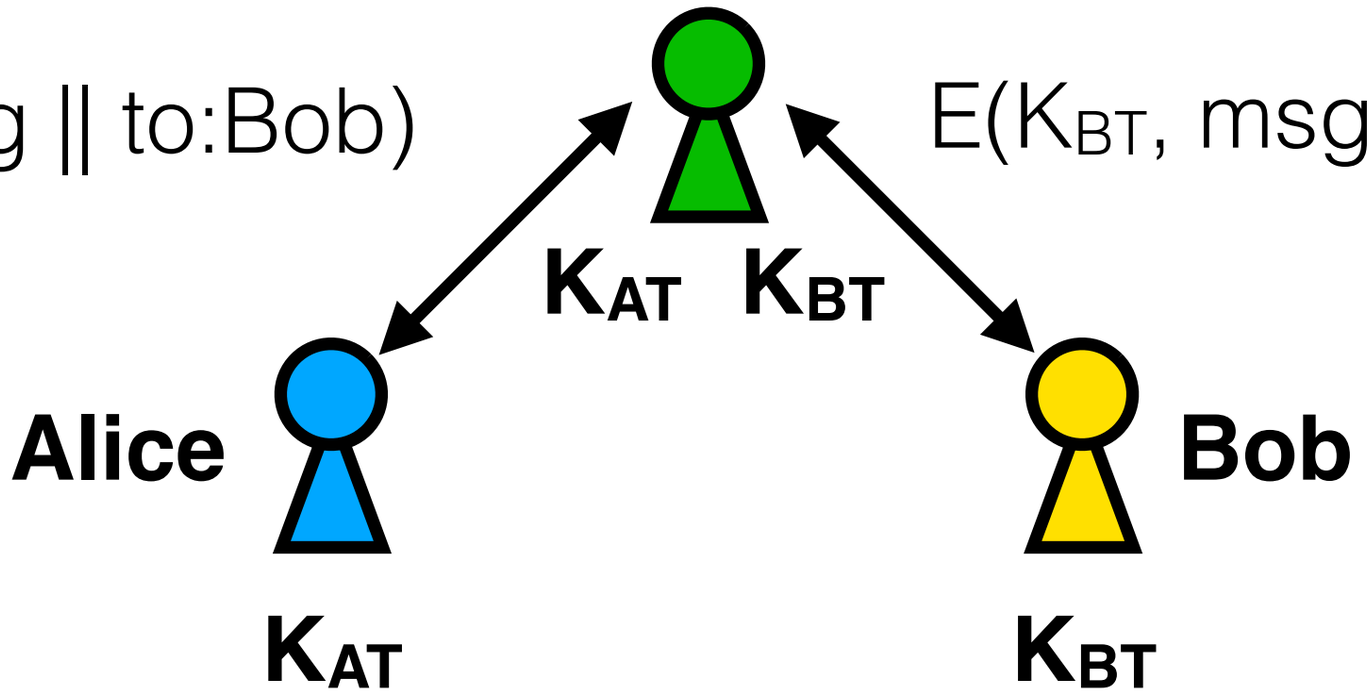
Just as “secure” meant nothing without an attack model, “trusted” means nothing without a **trust model**



What are we trusting Trent not to do?

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

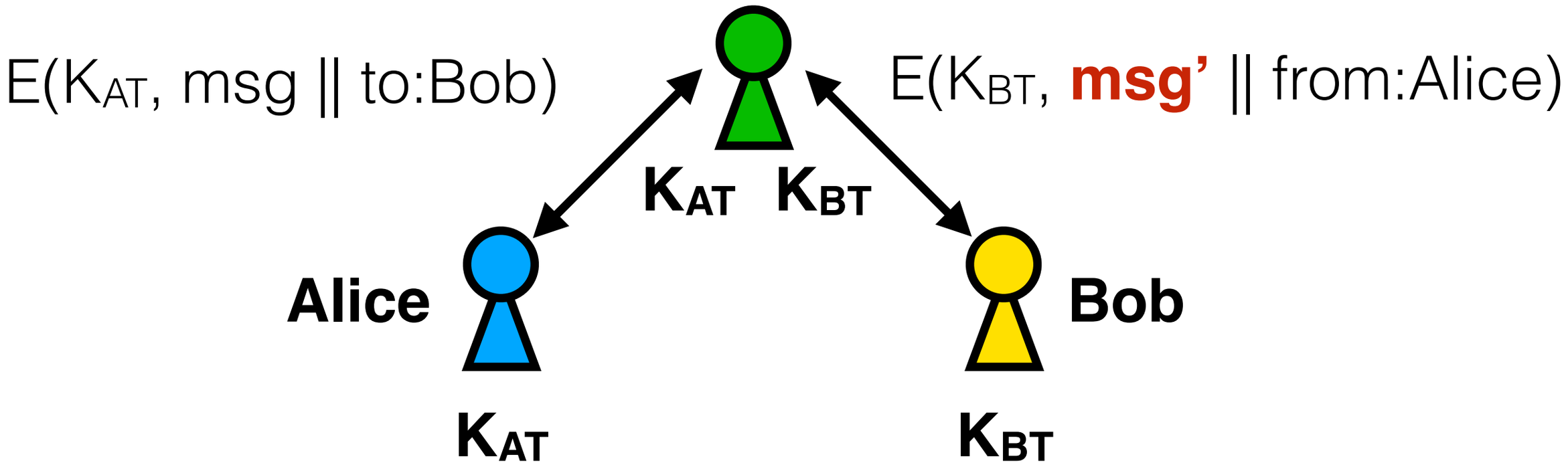
(Oh wow, “msg”!)



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**

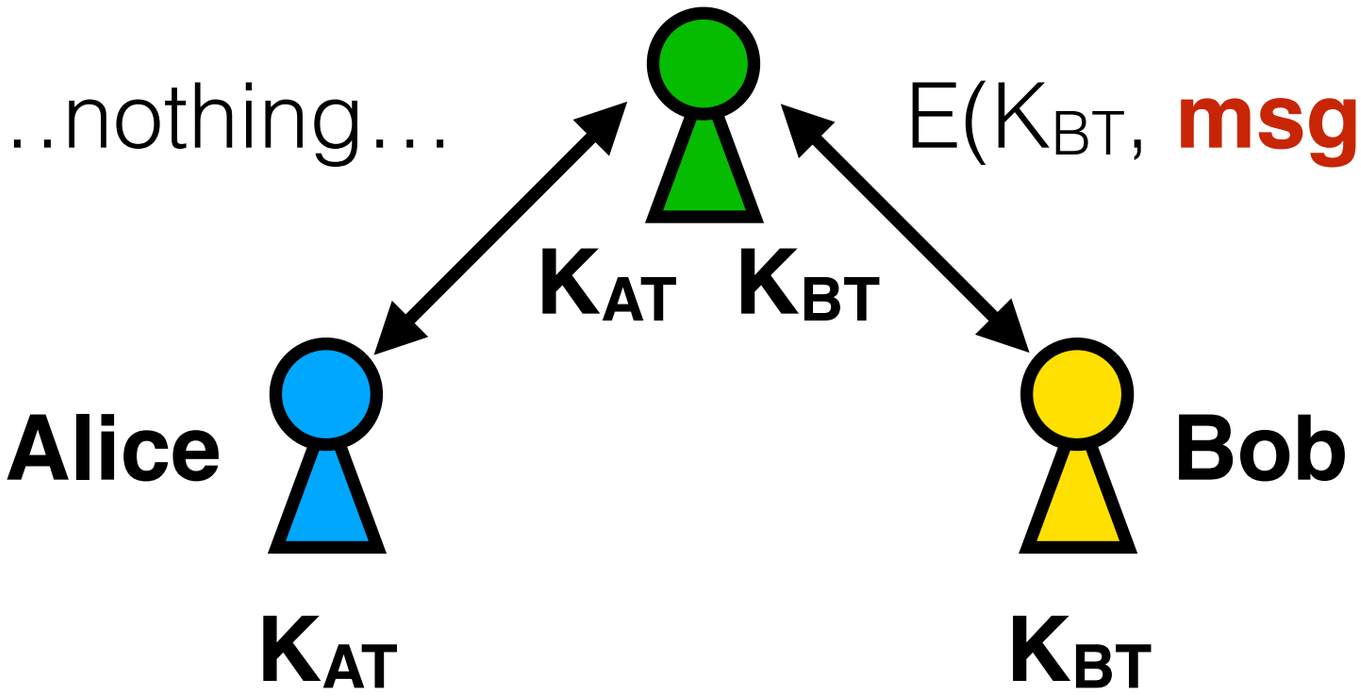


1. Do not *read* messages
2. Do not *alter* messages

What are we trusting Trent not to do?

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

…nothing…



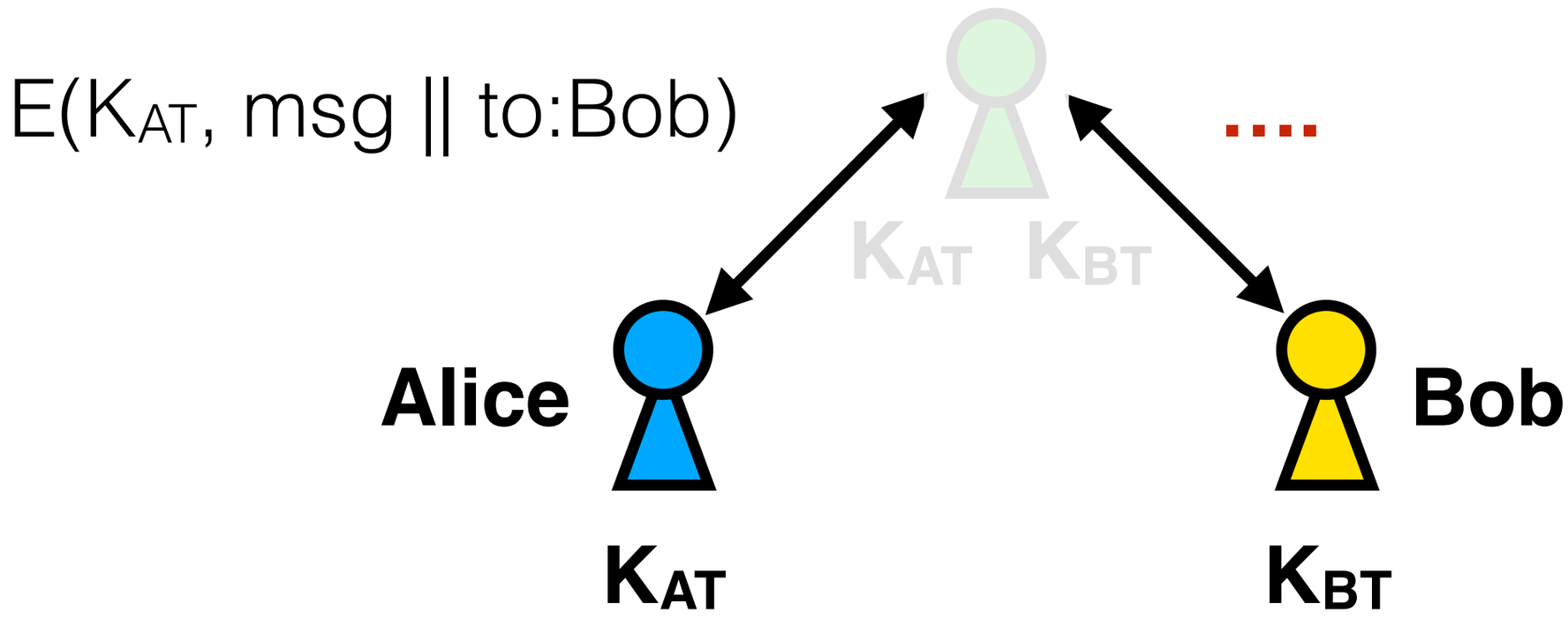
1. Do not *read* messages

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)

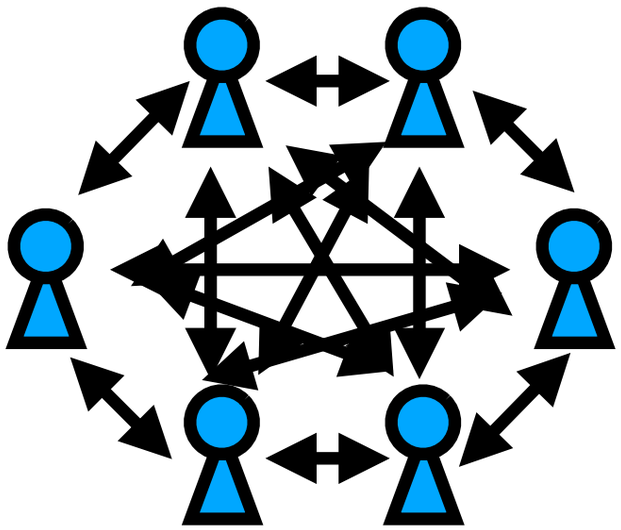
E() should approximate a one-way
trapdoor function: cannot invert
without access to SK

Protocols with public key encryption

Goal: deliver a confidential message

Symmetric key

Email / chat



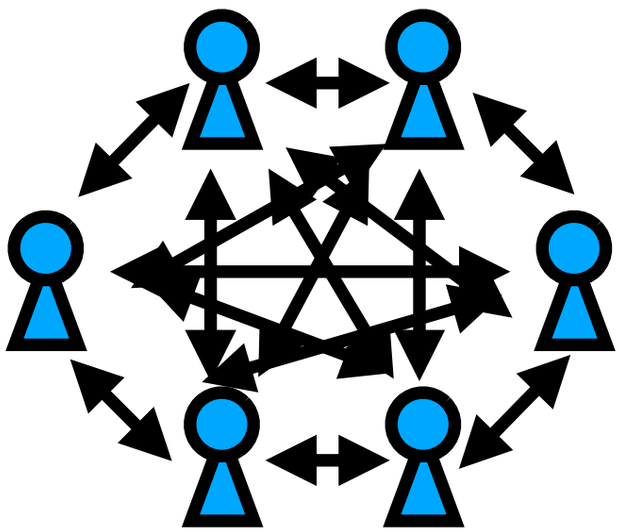
All-to-all: $O(N^2)$ key exchanges

Protocols with public key encryption

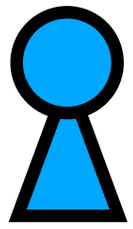
Goal: deliver a confidential message

Symmetric key

Email / chat



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



Generate public/private key pair (PK,SK)

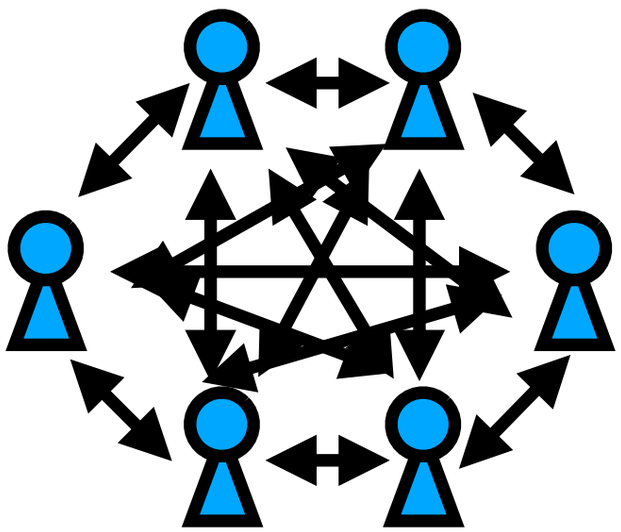
Annouce PK publicly
(on website, in newspaper, …)

Protocols with public key encryption

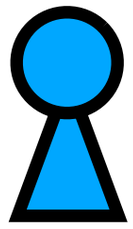
Goal: deliver a confidential message

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

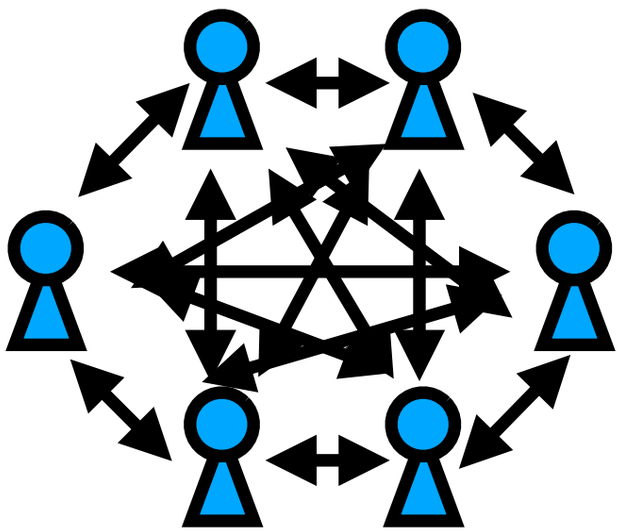
Send c = E(PK, msg)

Protocols with public key encryption

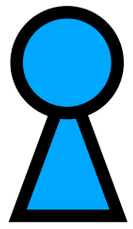
Goal: deliver a confidential message

Symmetric key

Email / chat



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

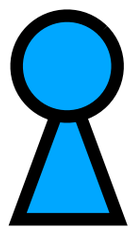


Generate public/private key pair (PK,SK)

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

Obtain PK

Send c = E(PK, msg)



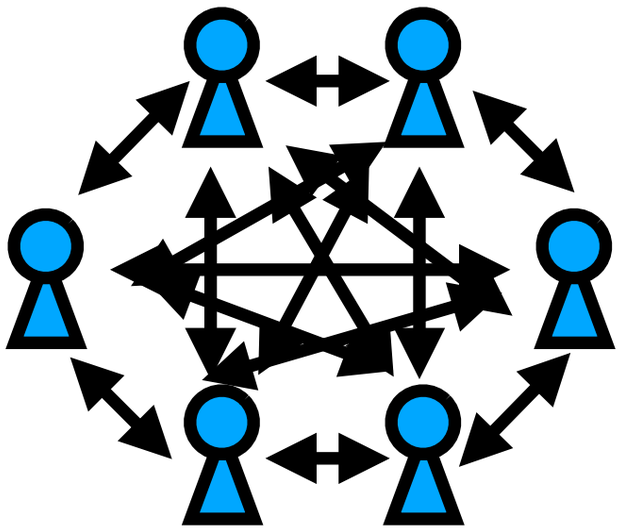
Decrypt D(SK, c) = msg

Protocols with public key encryption

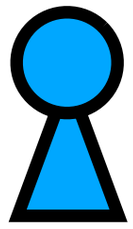
Goal: deliver a confidential message

Symmetric key

Email / chat



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

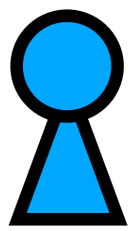


Generate public/private key pair (PK,SK)

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*

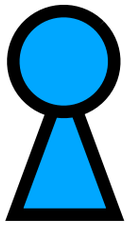
Like block ciphers, but there are not "modes" of public key encryption

Public key operations are *slooooow!*

Symmetric key operations are fast

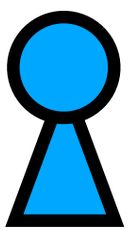
Hybrid encryption

Hybrid encryption



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

Hybrid encryption



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

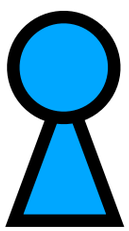
Obtain PK

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

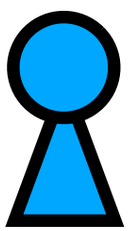
Obtain PK

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

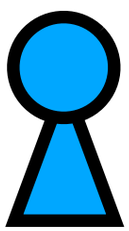
Obtain PK

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

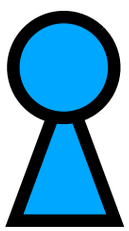
Generate *symmetric* key K

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

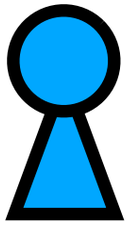
Obtain PK

Generate *symmetric* key K

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

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

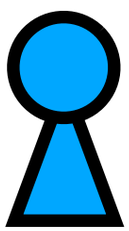
Send $c_K \,||\, c_{msg}$



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

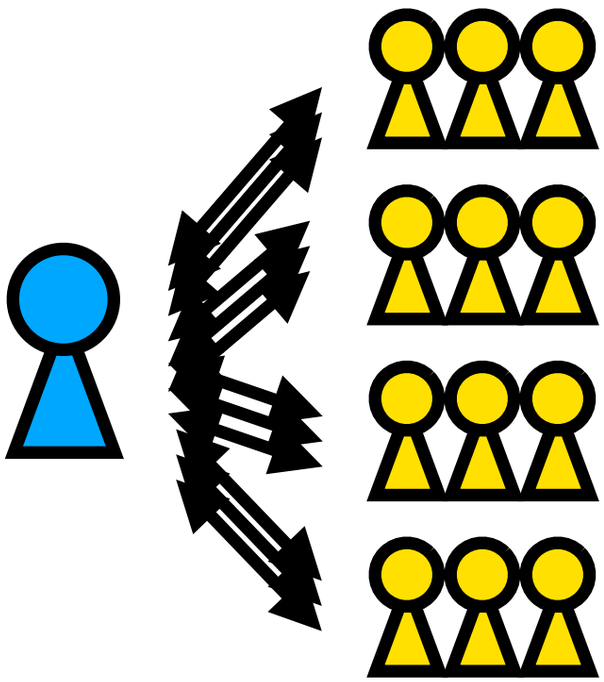
The speed and arbitrary message length of symmetric key

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

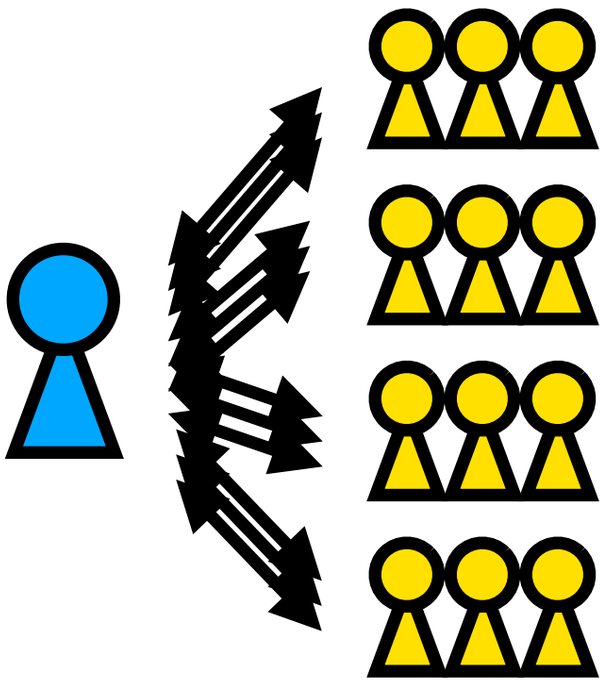
Same as with MACs: even after a chosen plaintext attack, the attacker cannot demonstrate an existential forgery

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)

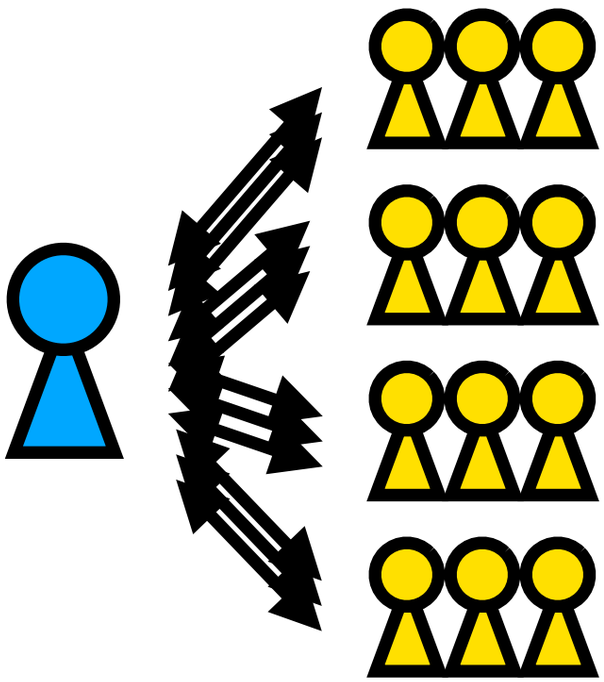
Annouce PK publicly (on website, in newspaper, …)

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, …)

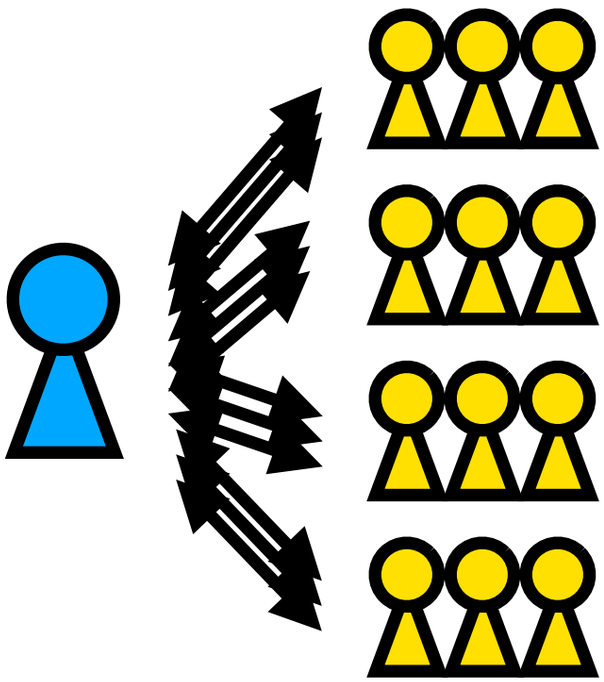
Compute sig = Sgn(SK, msg)

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)

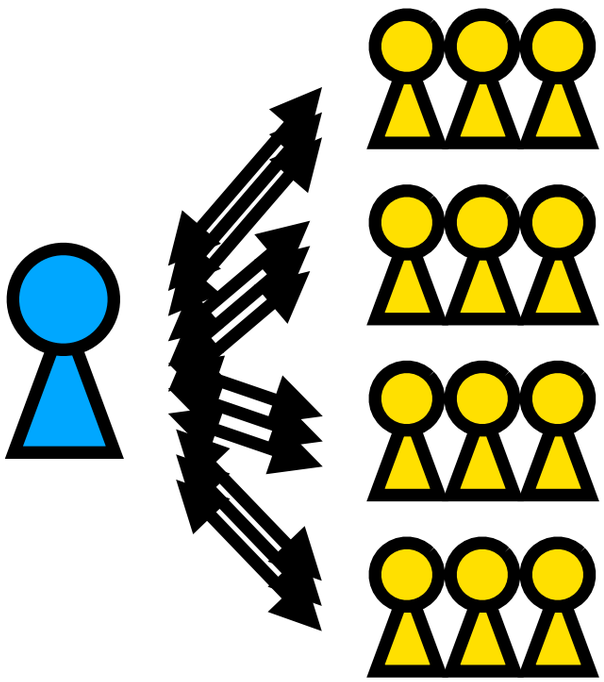
Publish msg || sig

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

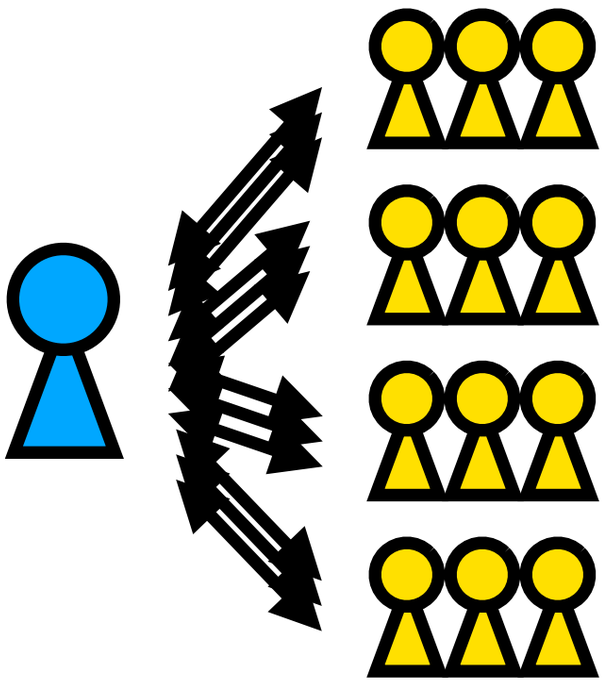
can now go offline!

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)

Back to authentication

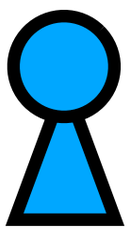
How can we know it was really  who posted PK?

Back to authentication

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

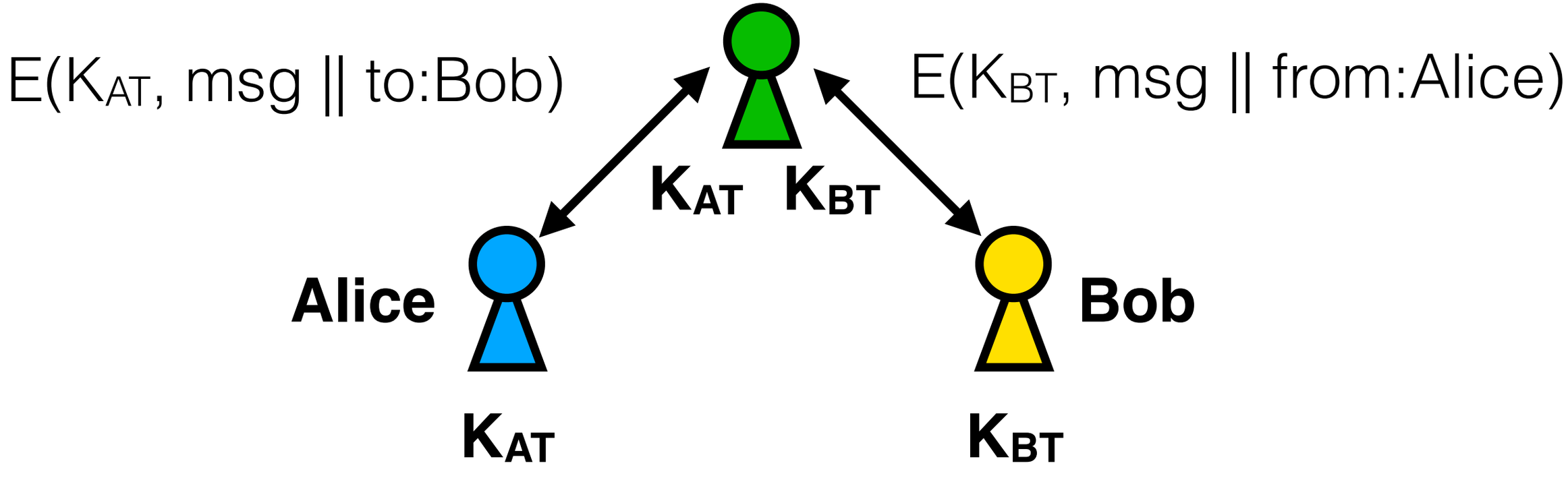
How can we know it was really  who posted PK?

Back to authentication

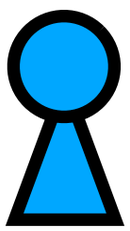


Generate public/private key pair (PK,SK); publicize PK

How can we know it was really  who posted PK?

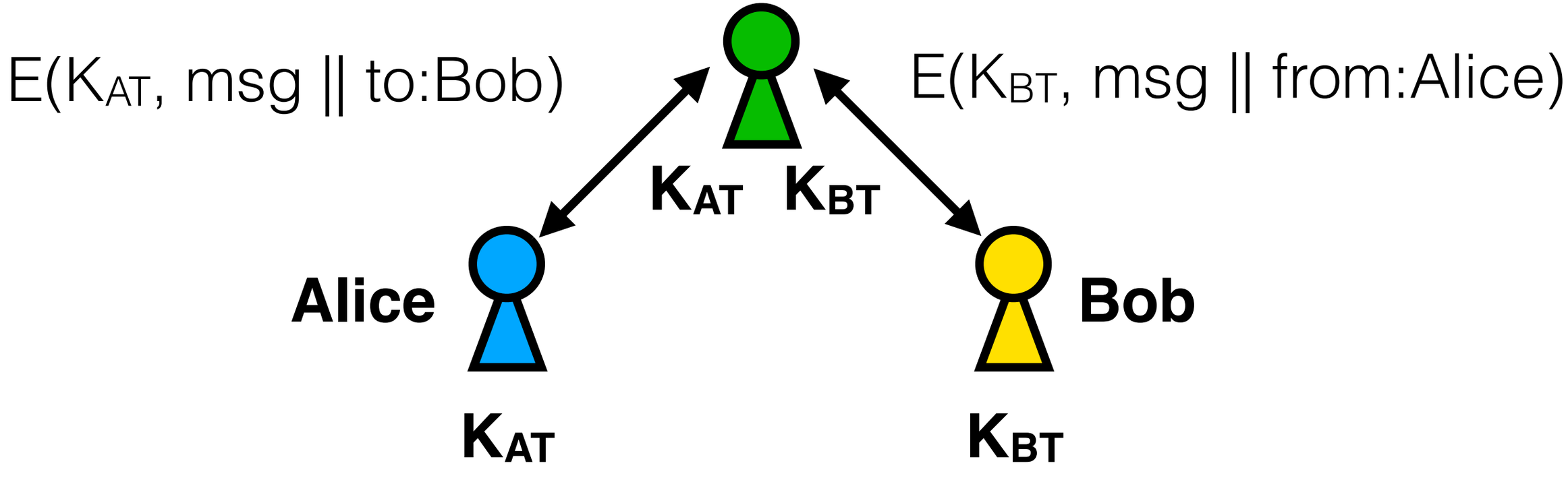


Back to authentication



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)**

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

Alice 

Bob 

Authentication with public keys

Trent  **(PK_T, SK_T)**

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

Alice  **PK_T**

Bob  **PK_T**

Authentication with public keys

Trent  **(PK_T, SK_T)**

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

Alice 

PK_T
(PK_A, SK_A)

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

Bob 

PK_T

Authentication with public keys

(PK_T, SK_T)

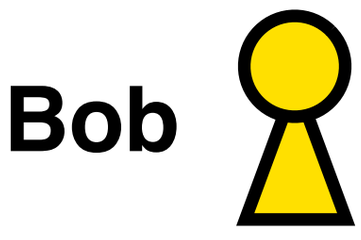
1. Trent's public key is widely disseminated (pre-installed in browsers/operating systems)

Trent vets Alice

PK_T

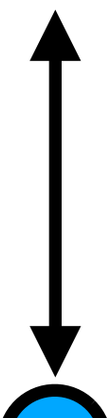
(PK_A, SK_A)

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

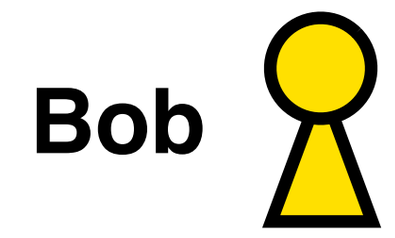


PK_T

Authentication with public keys



Trent vets Alice



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"

Authentication with public keys

Trent vets Alice

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

(PK_T, SK_T)

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

Trent vets Alice

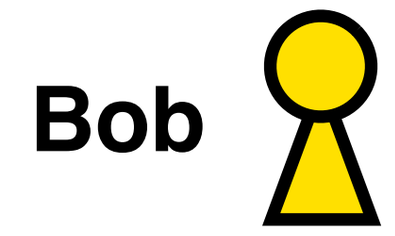
PK_T

(PK_A, SK_A)

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

Alice = PK_T

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

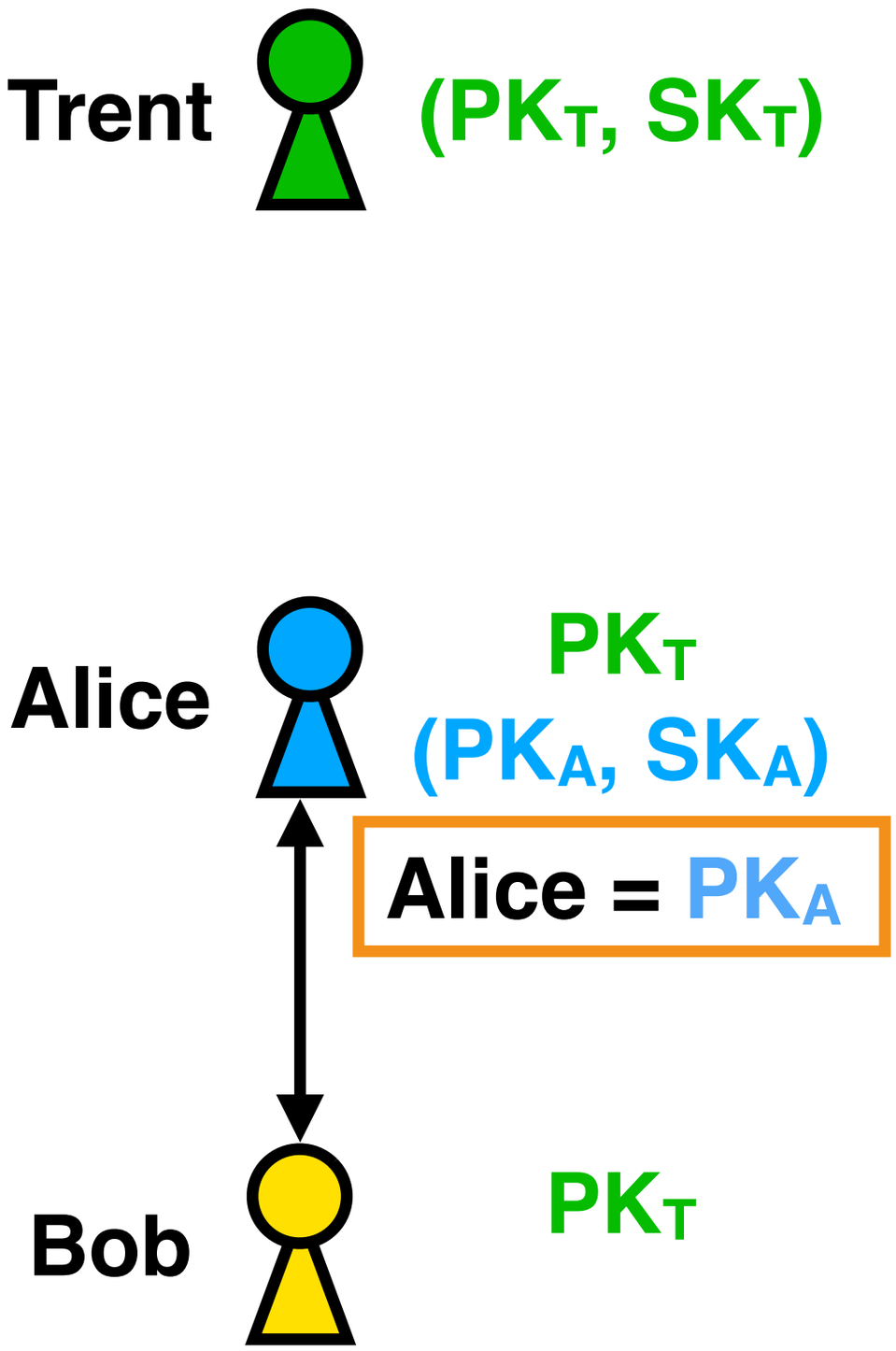


PK_T

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

This message + sig = Certificate

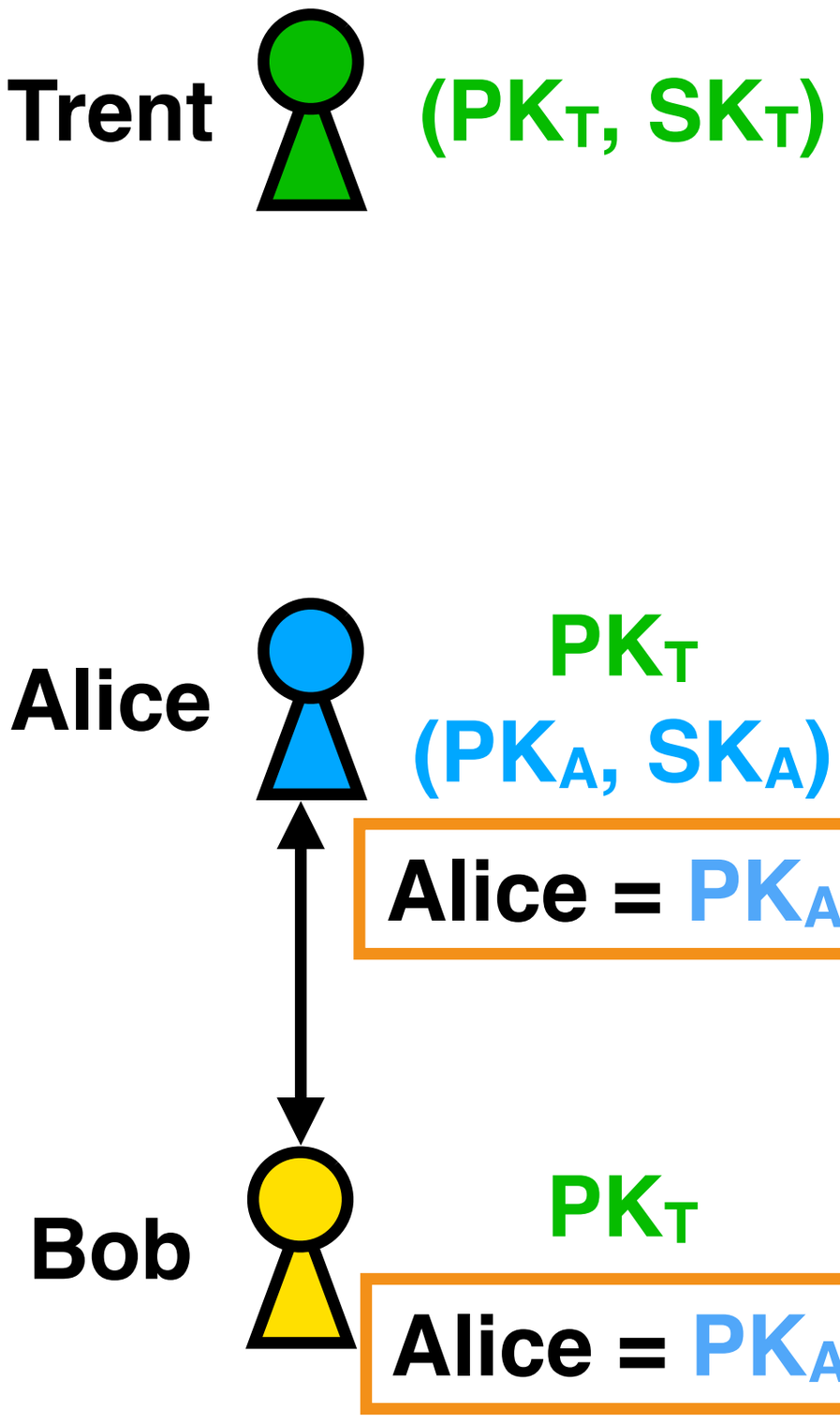
Authentication with public keys



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

If Bob trusts Trent, then Bob trusts that he knows Alice's public key is **PK$_A$**

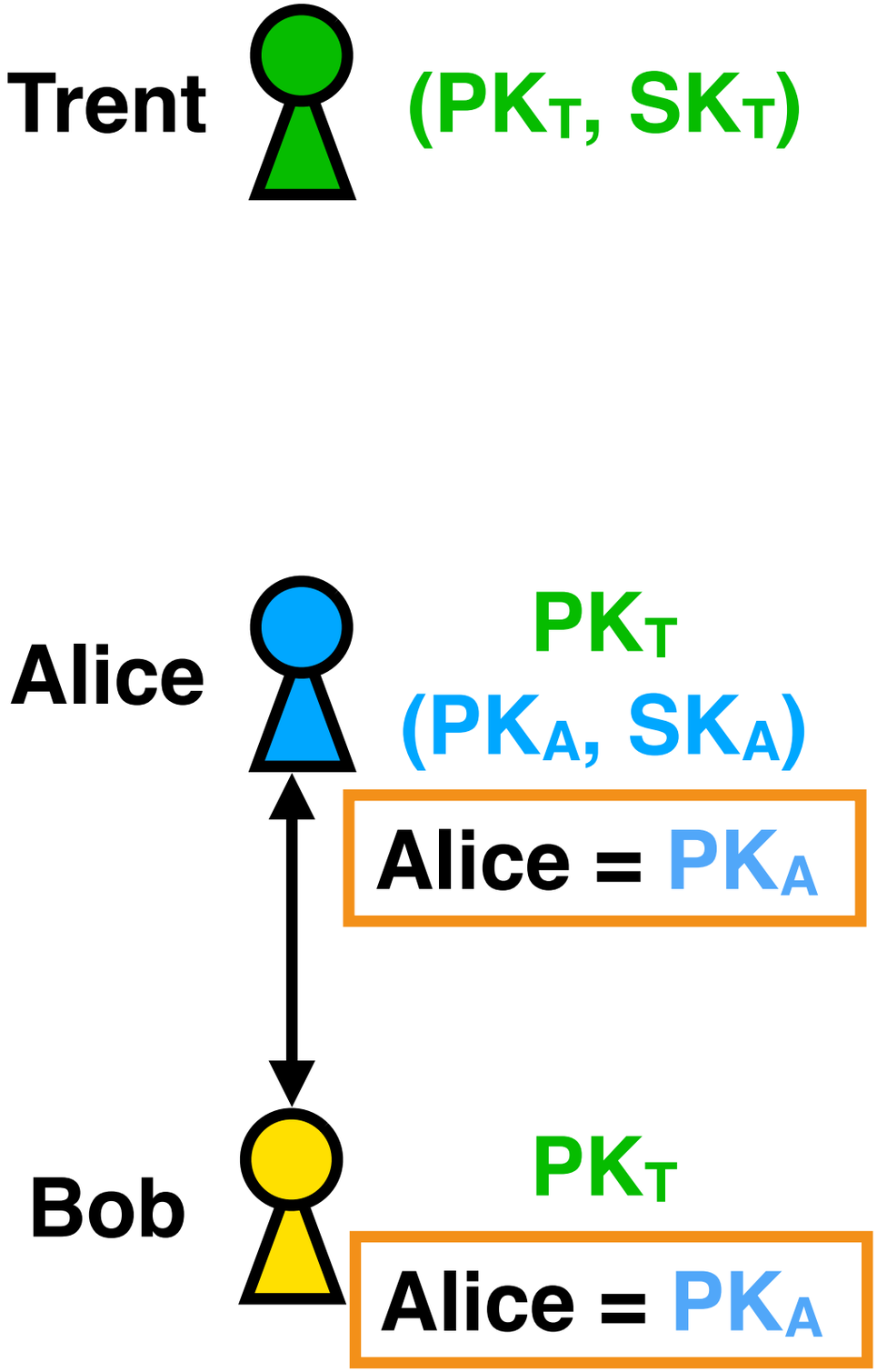
Authentication with public keys



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

If Bob trusts Trent, then Bob trusts that he knows Alice's public key is PK_A

Authentication with public keys

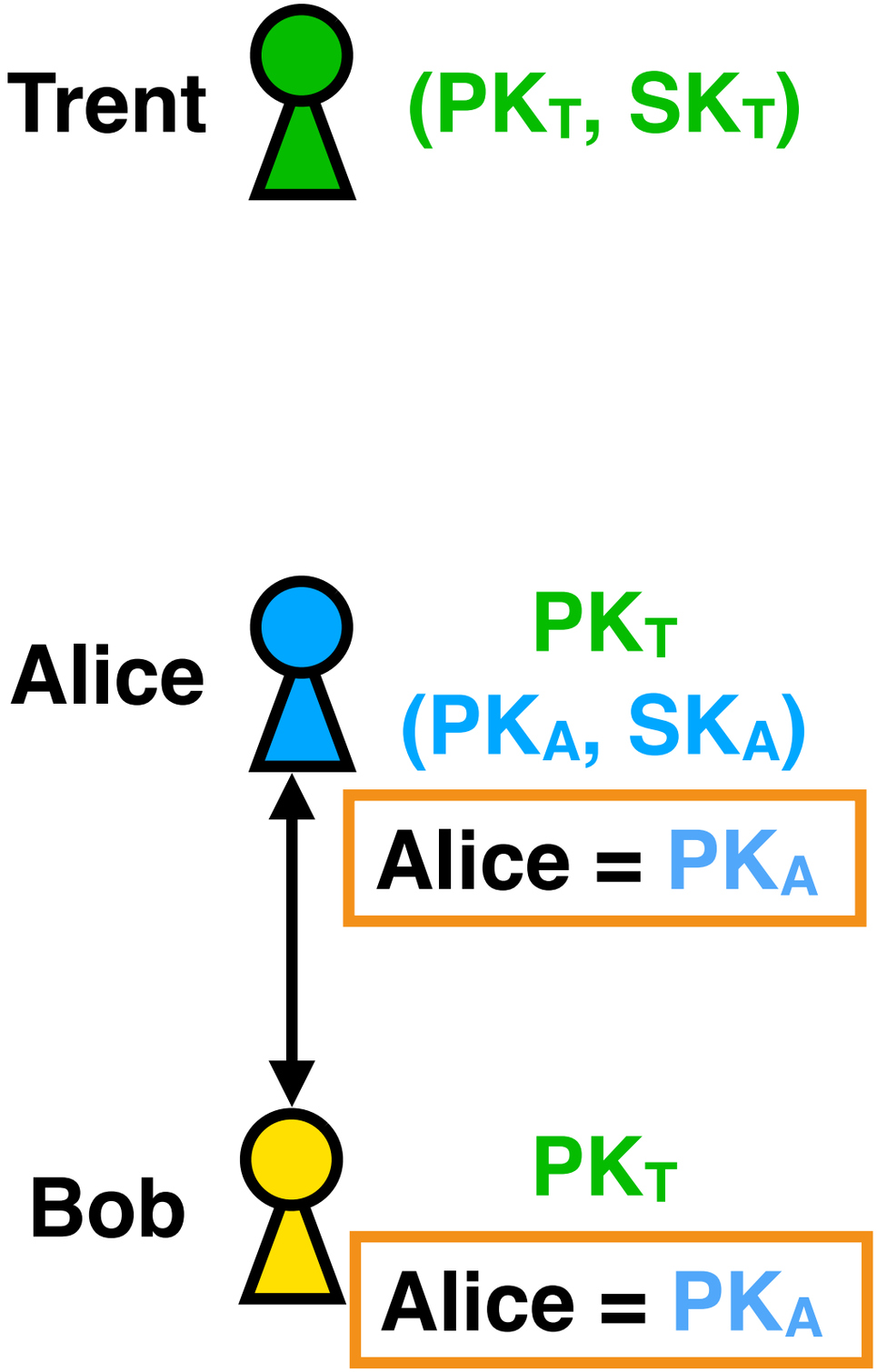


4. Alice makes her certificate publicly available (or Bob simply asks for it)

5. Bob verifies the certificate using PK_T

If Bob trusts Trent, then Bob trusts that he knows Alice's public key is PK_A

Authentication with public keys



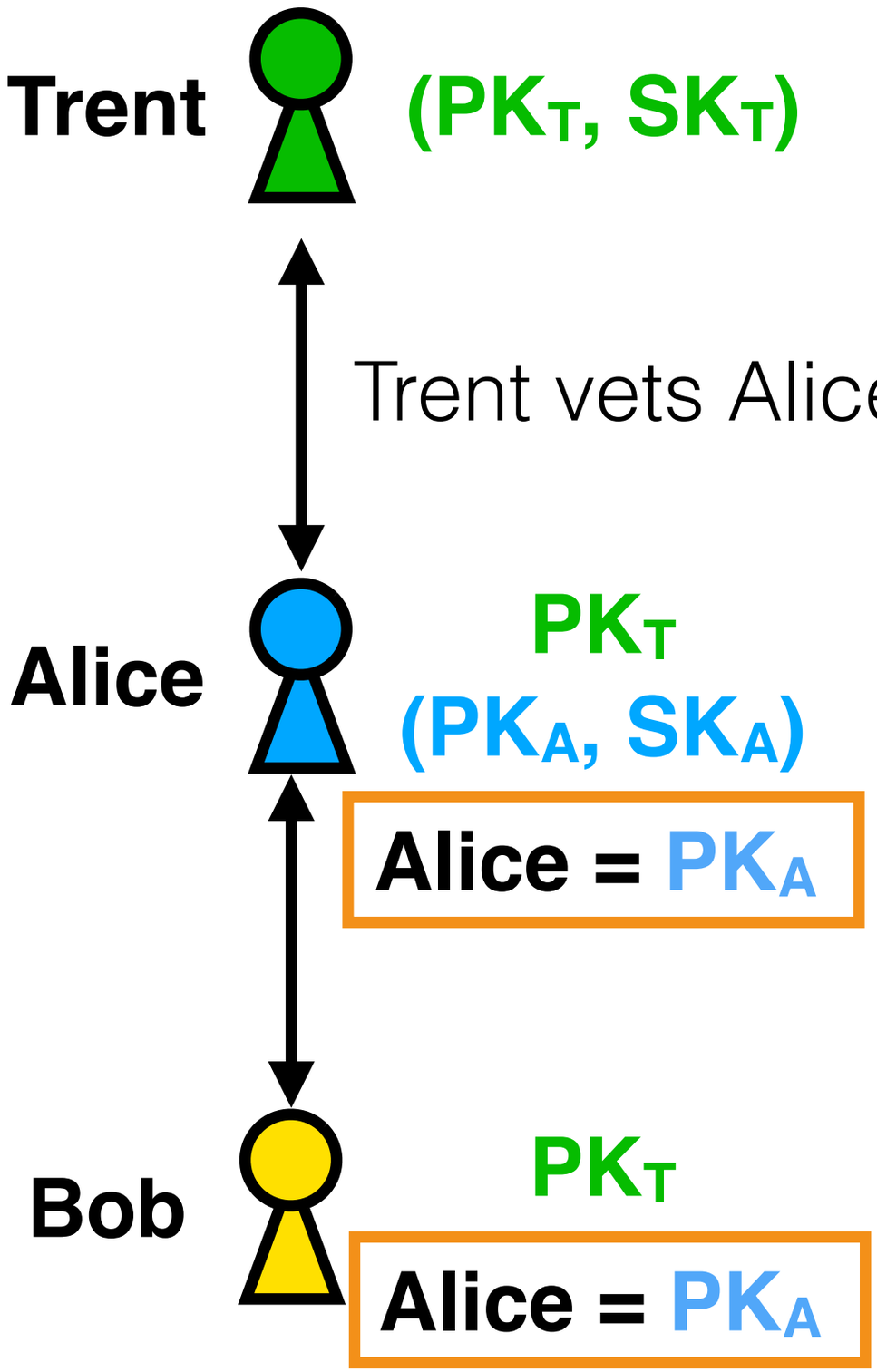
4. Alice makes her certificate publicly available (or Bob simply asks for it)

5. Bob verifies the certificate using PK_T

If Bob trusts Trent, then Bob trusts that he knows Alice's public key is PK_A

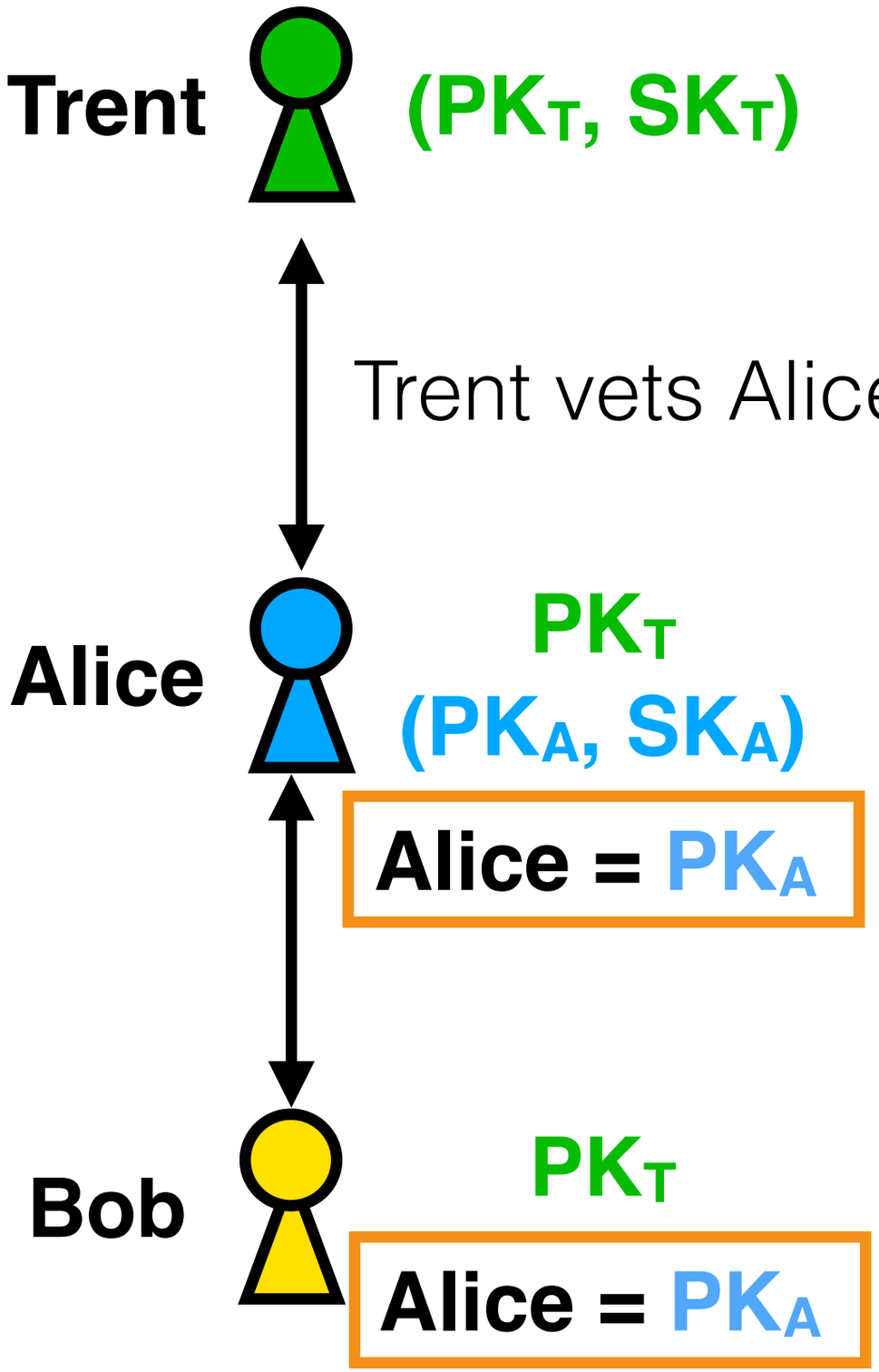
6. Bob (via hybrid encryption) sends a message to Alice using her public key PK_A

Authentication with public keys



Properties

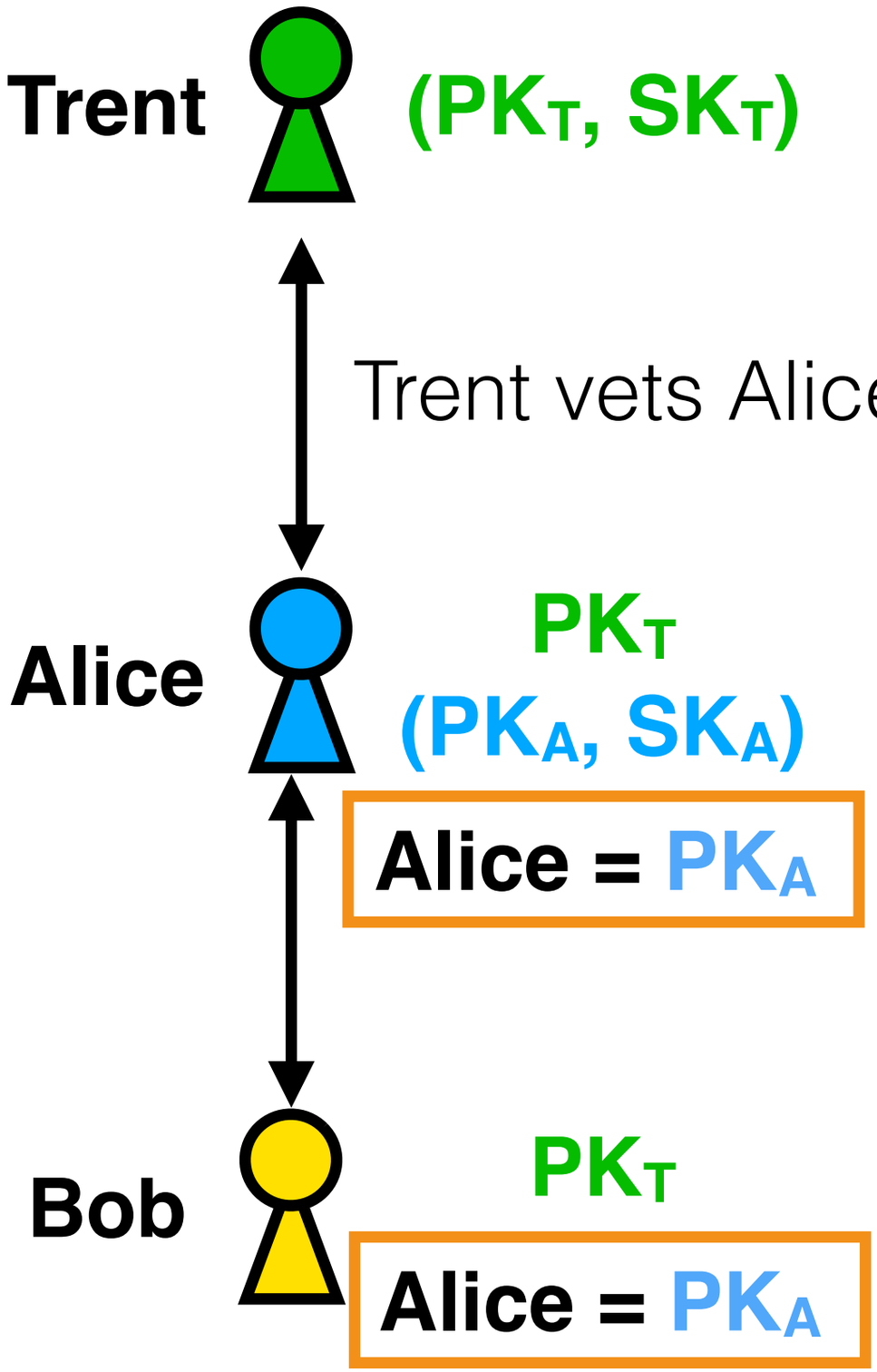
Authentication with public keys



Properties

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

Authentication with public keys

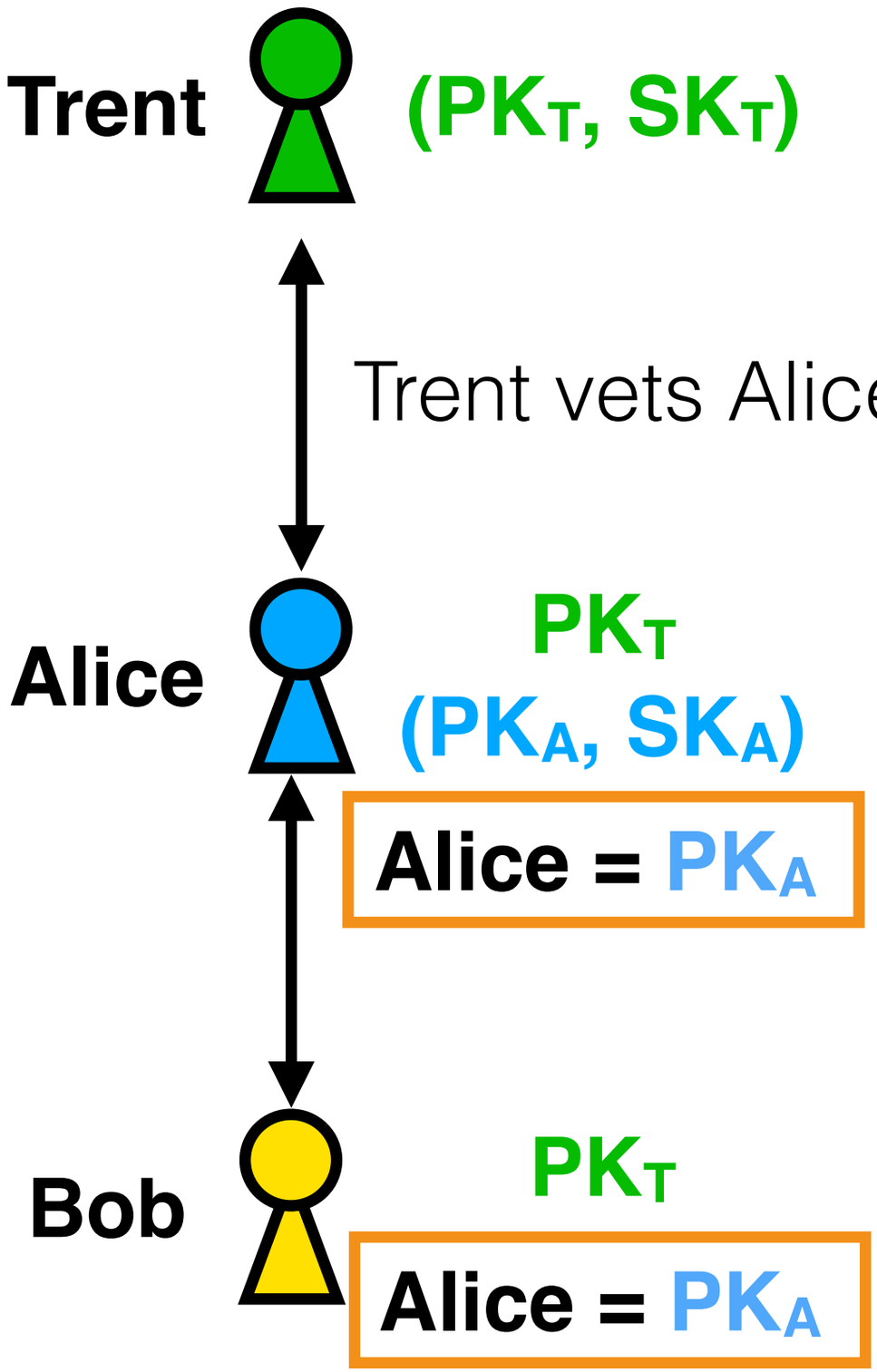


Properties

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

Communication needn't go through Trent

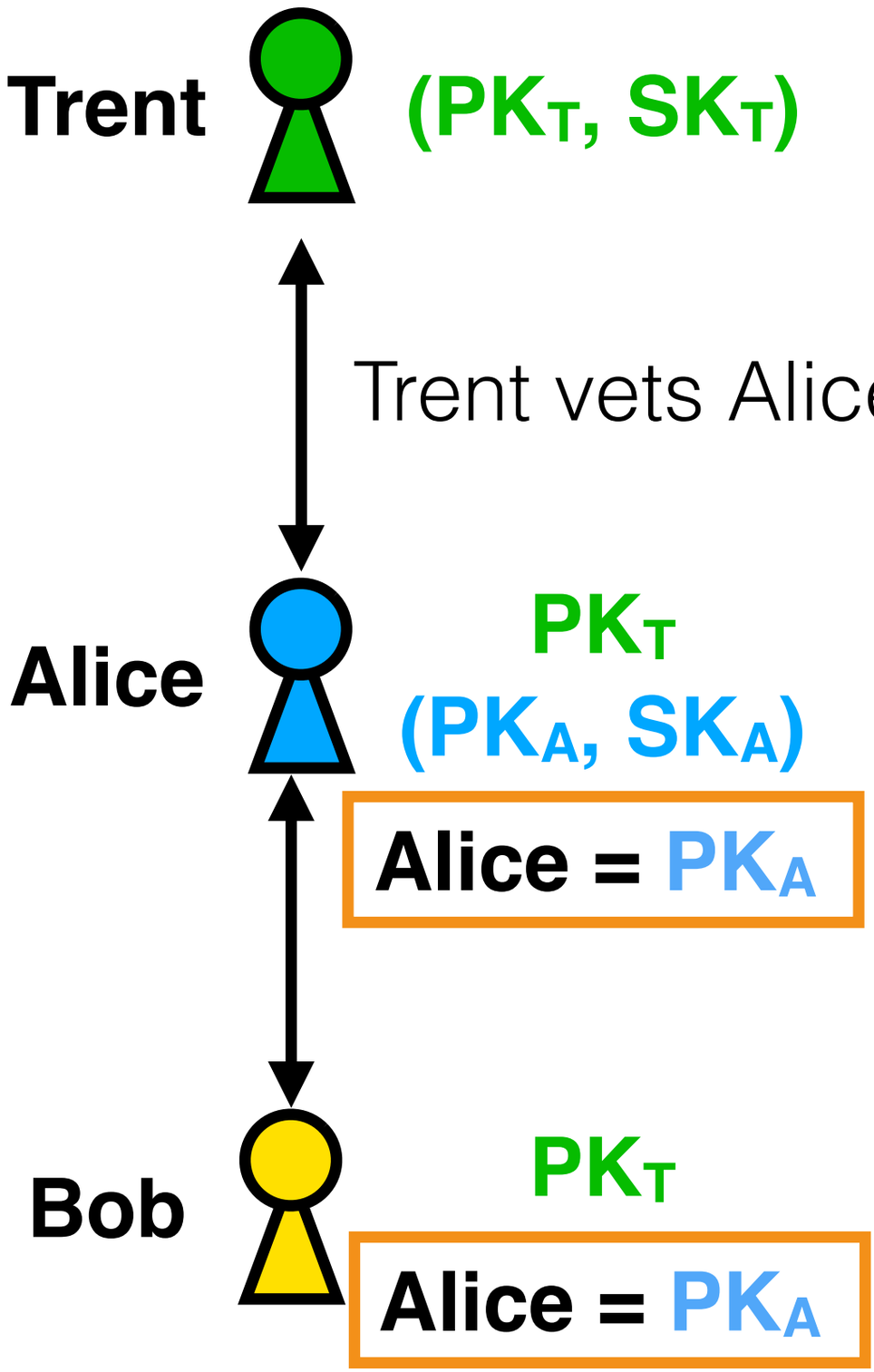
Authentication with public keys



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*

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**):

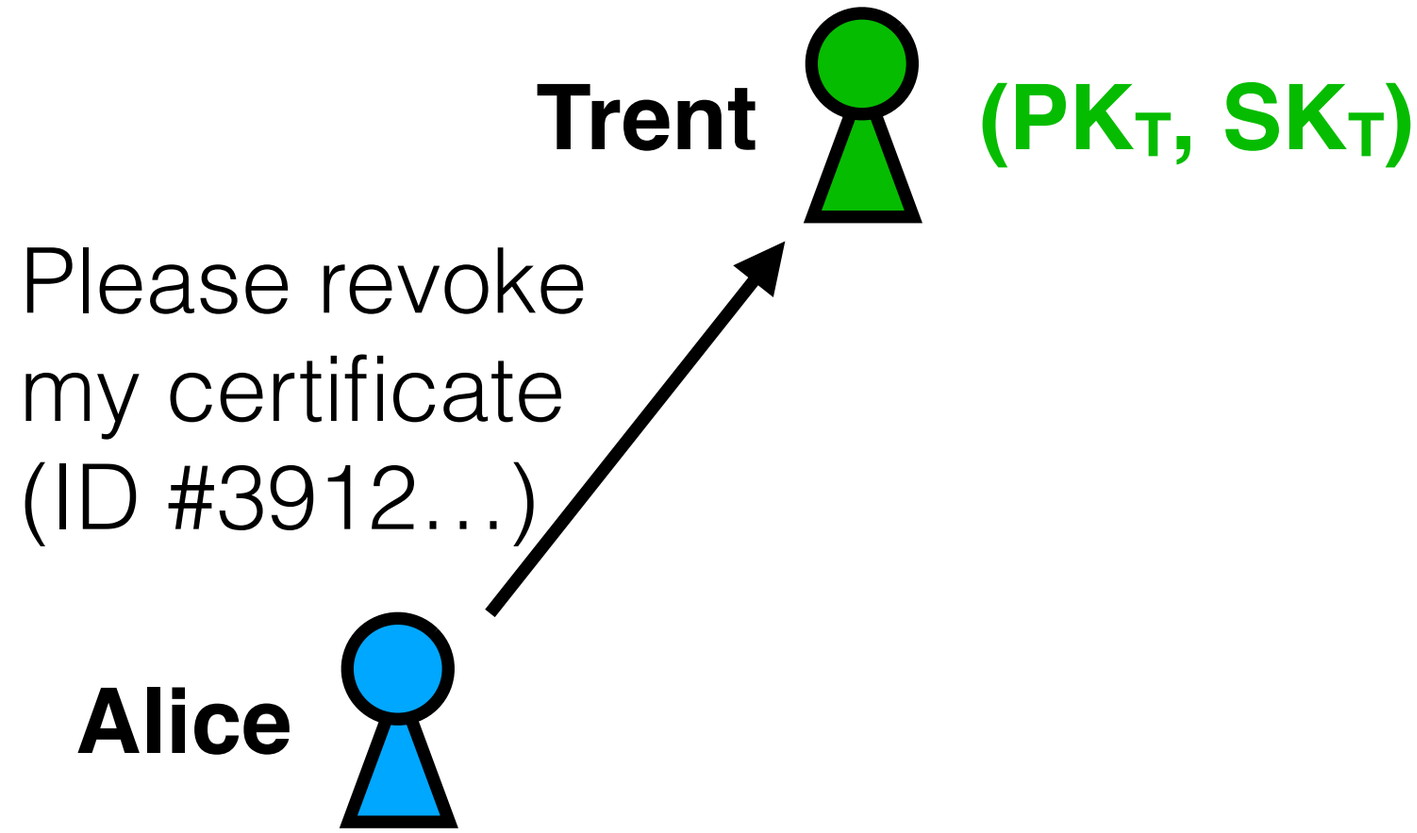
"The owner of the secret key corresponding to **PK_A** is Alice"

This message + sig = Certificate

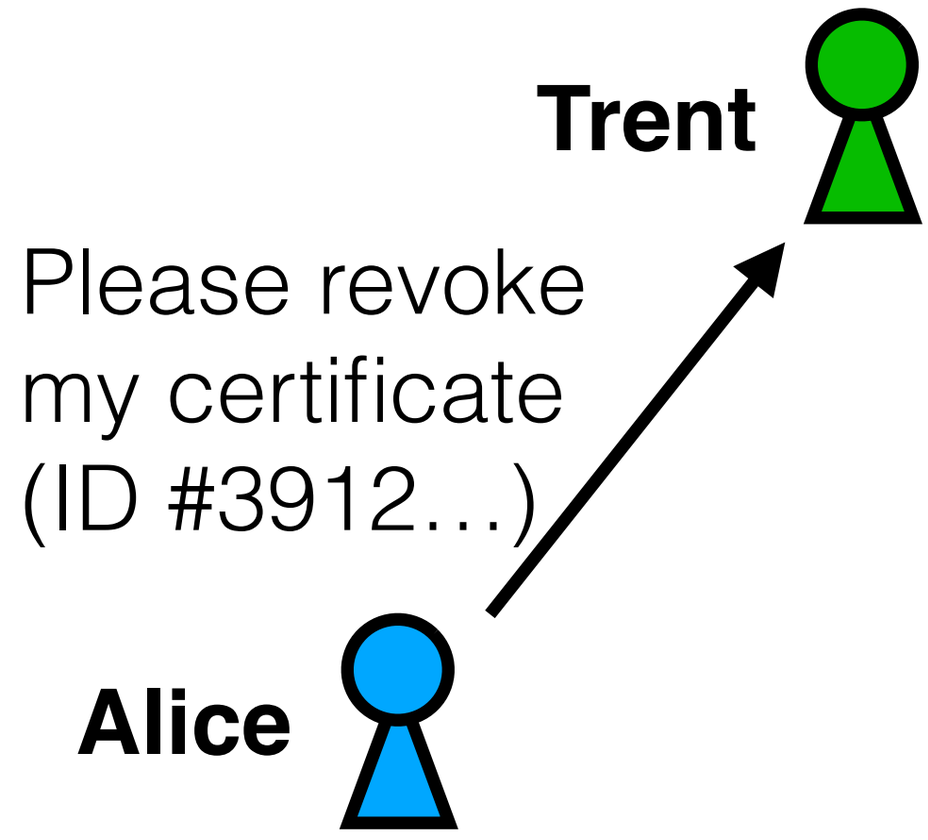
Put another way:
"The only person who knows **SK_A** is Alice"

What happens if Alice's key gets compromised?
(Stolen, accidentally revealed, …)

Certificate revocation



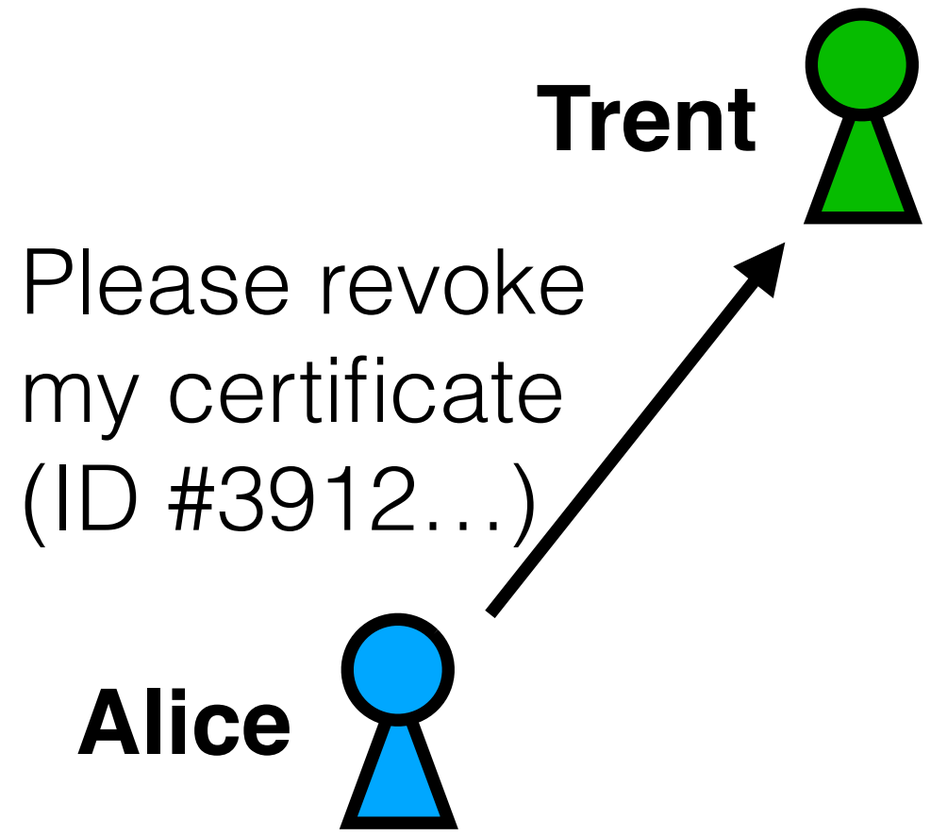
Certificate revocation



Trent *signs* a message (with SK_T):

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

Certificate revocation



Trent *signs* a message (with SK_T):

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

This message + sig = revocation

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

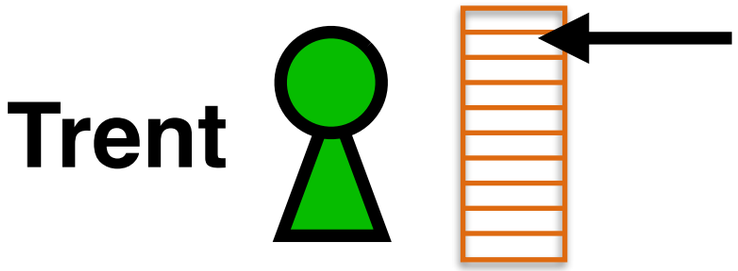
Bob

Bob obtains revocation information

Obtaining revocation data

Certificate Revocation Lists (CRLs)

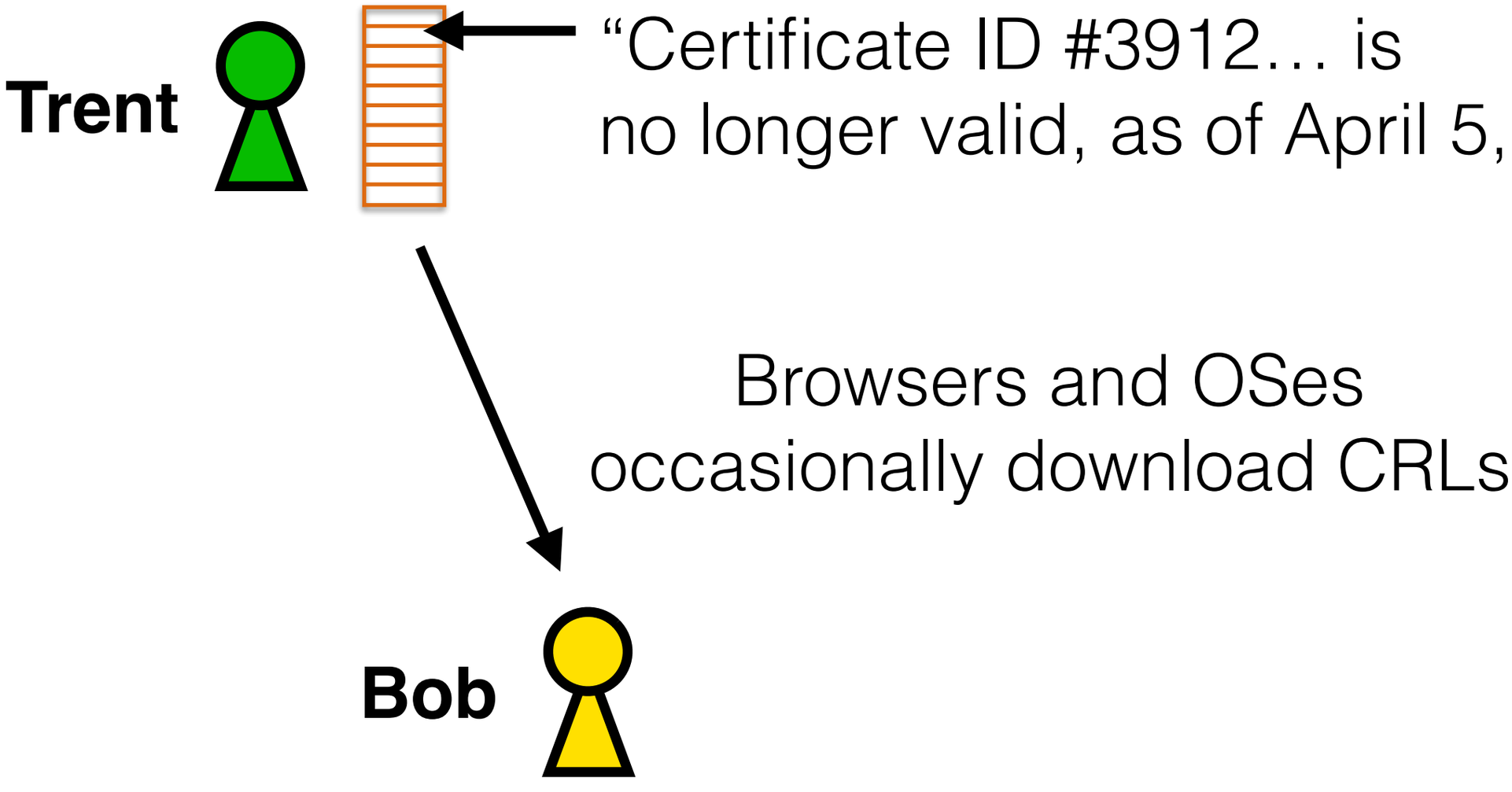
A (often large) signed list of revocations



Obtaining revocation data

Certificate Revocation Lists (CRLs)

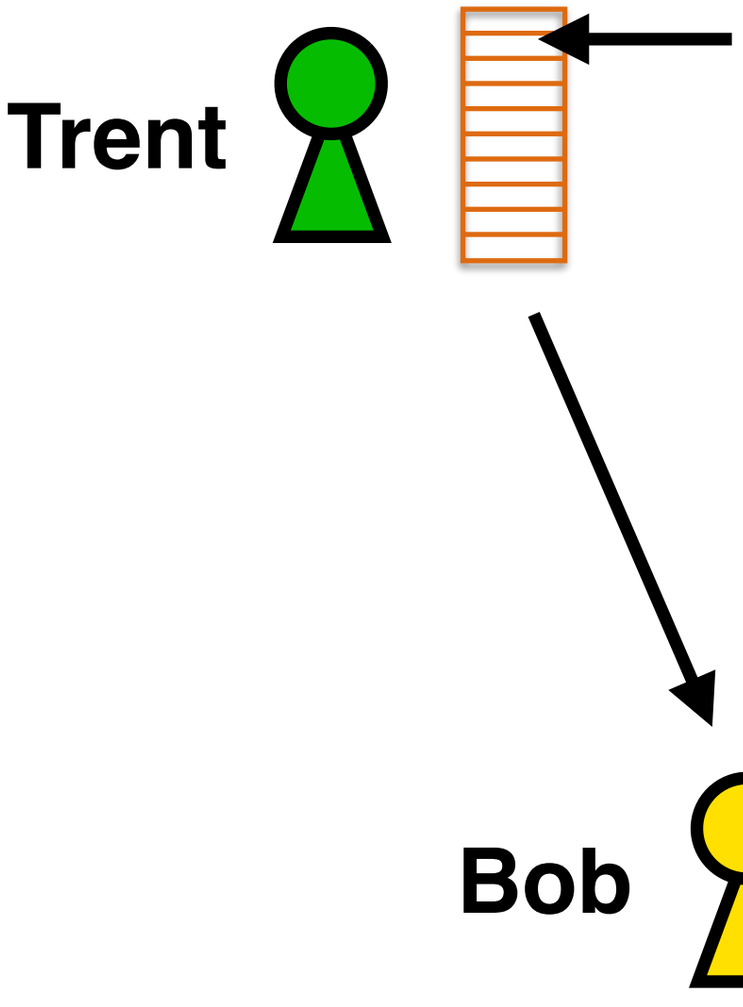
A (often large) signed list of revocations



Obtaining revocation data

Certificate Revocation Lists (CRLs)

A (often large) signed list of revocations

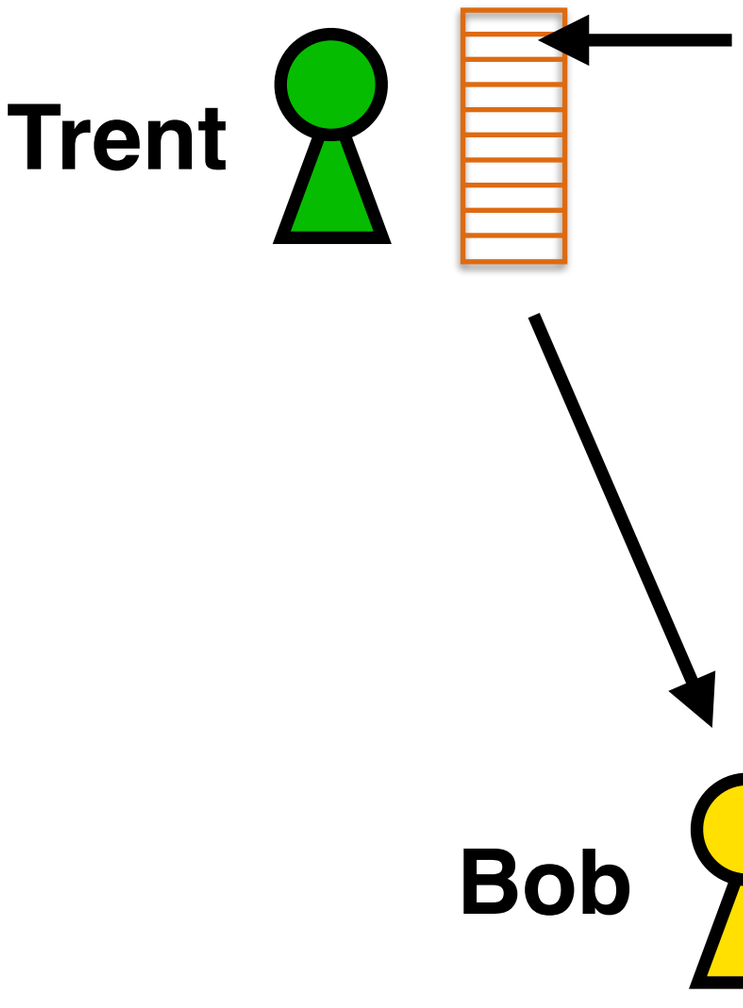


Disincentive: CRLs can be large, so it takes time & bandwidth

Obtaining revocation data

Certificate Revocation Lists (CRLs)

A (often large) signed list of revocations



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)

Obtaining revocation data

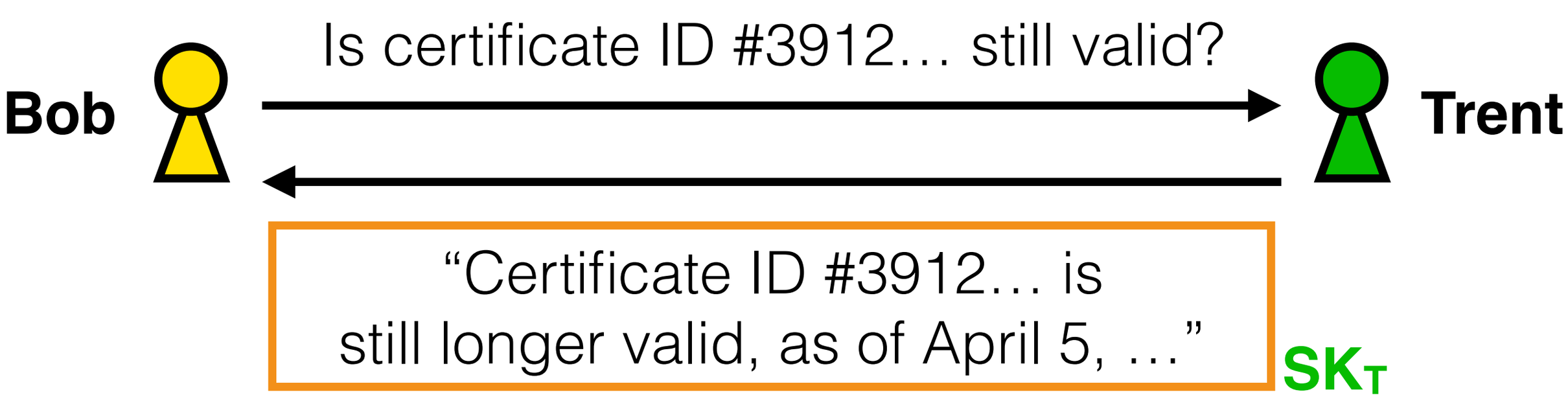
Online Certificate Status Protocol (OCSP)

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

Obtaining revocation data

Online Certificate Status Protocol (OCSP)

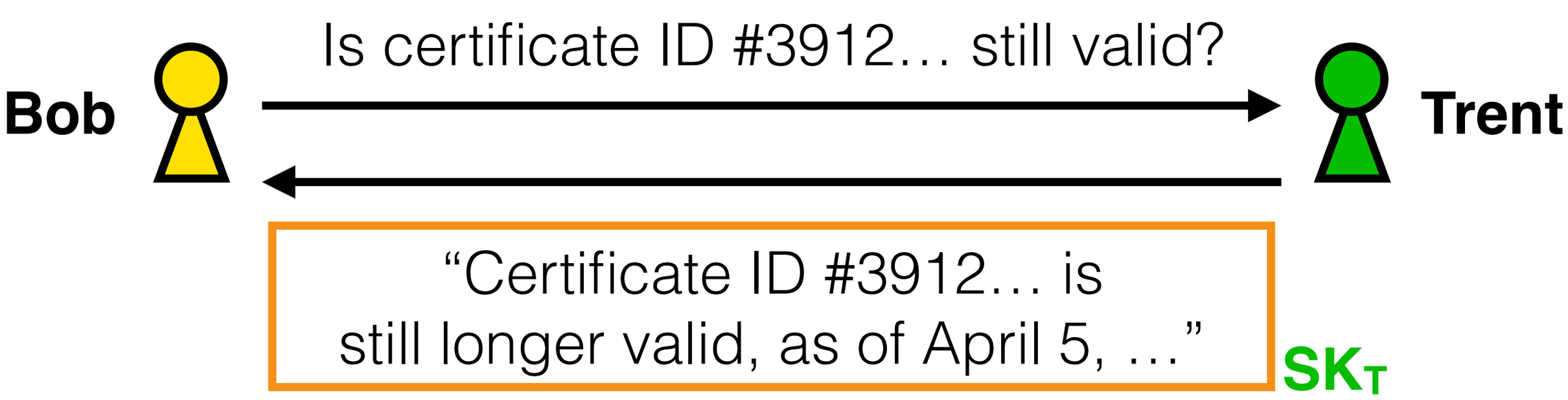
Browsers and OSes perform OCSP checks on-demand (when verifying the certificate)



Obtaining revocation data

Online Certificate Status Protocol (OCSP)

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

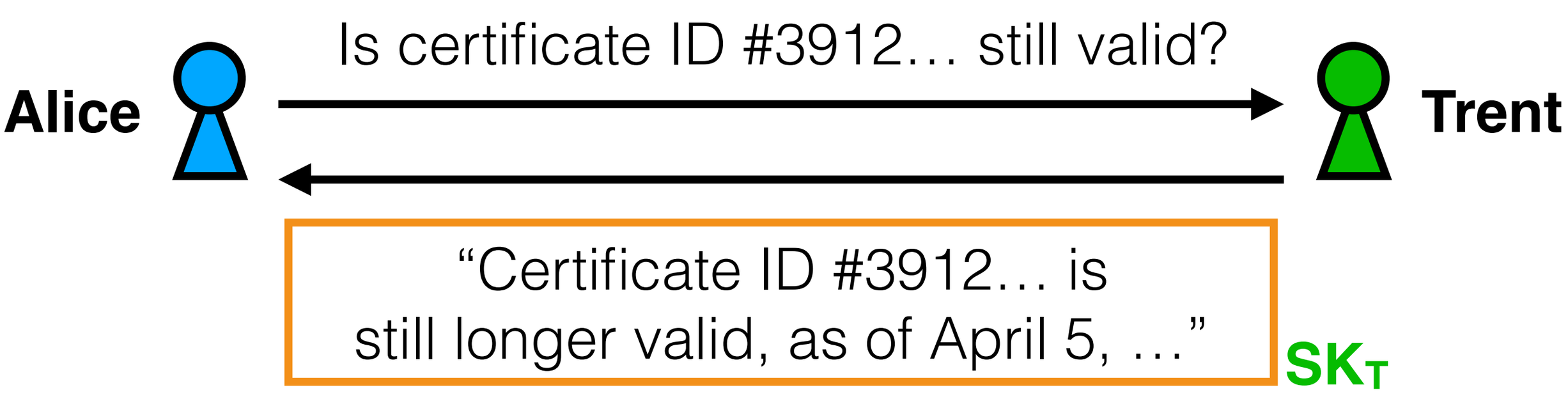


Disincentive: Still delays the initial validation of the certificate (can increase webpage load time)

Obtaining revocation data

OCSP Stapling

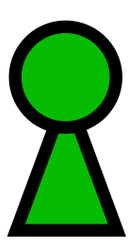
Websites issue OCSP requests,
include responses in initial handshake



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

Certificate revocation responsibilities

Alice's responsibility:
Request revocations

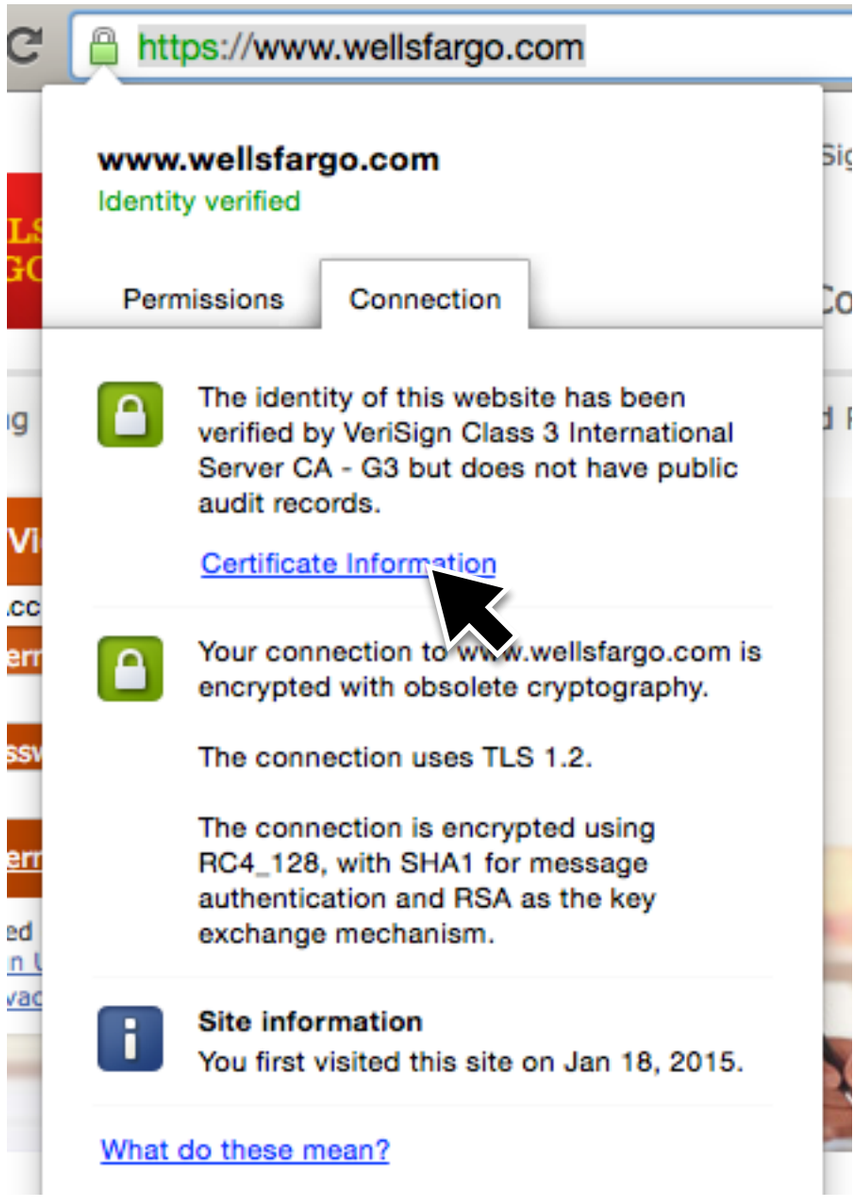


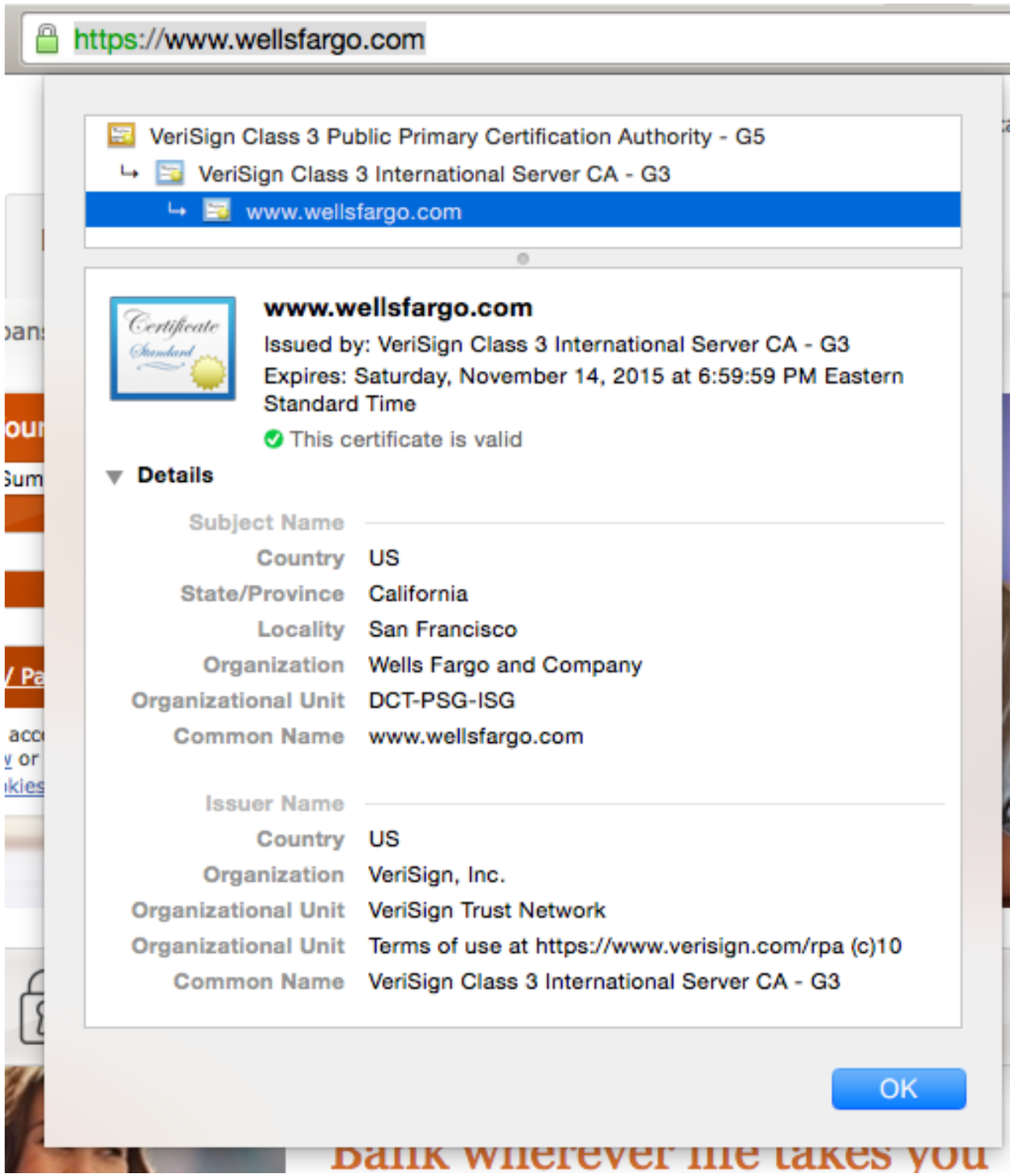
Trent's responsibility:
Make revocations publicly available

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)

Verifying certificates

“I’m because says so”

Verifying certificates

“I’m because says so”

“I’m because says so”

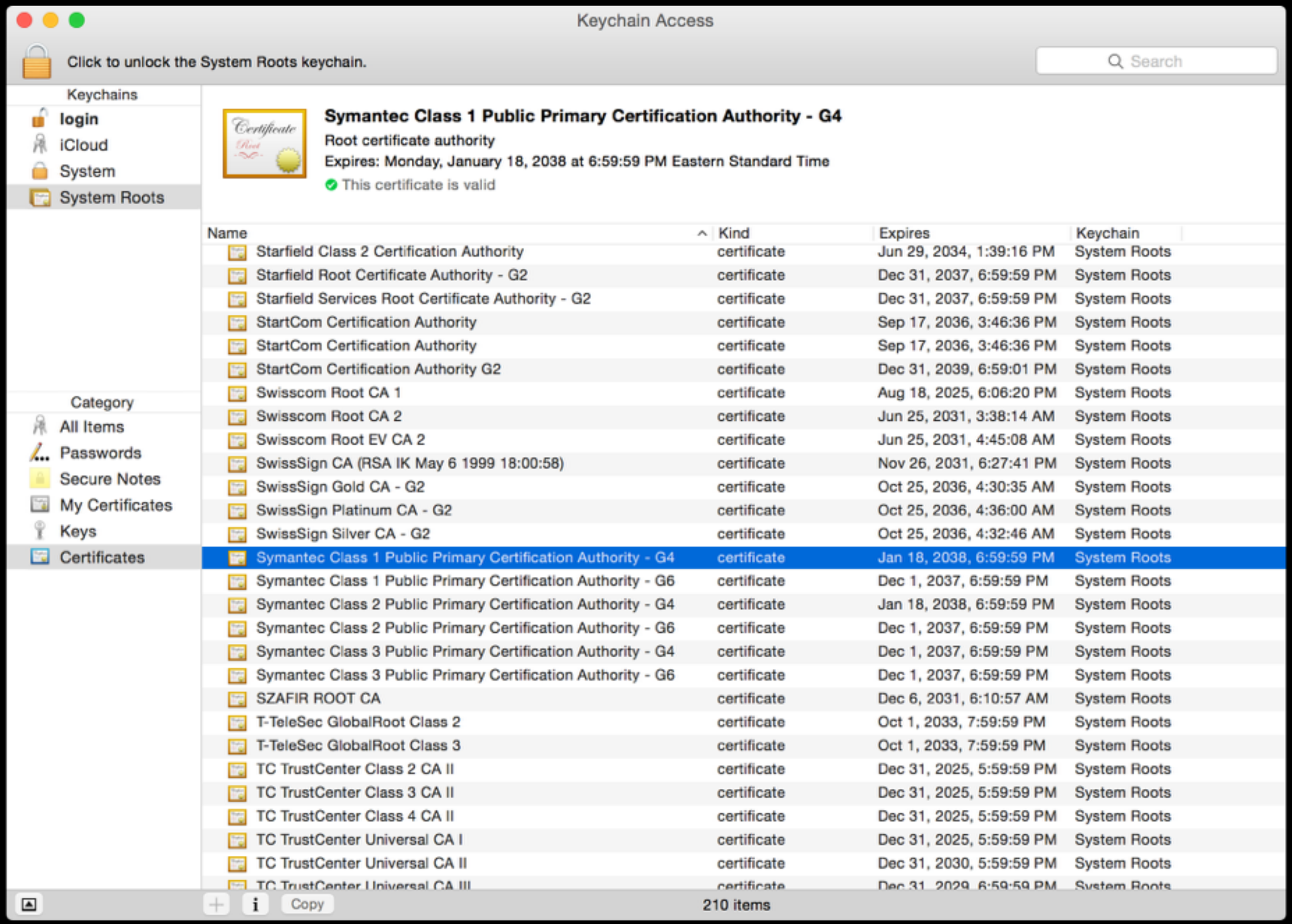
Verifying certificates

"I'm because I say so!"

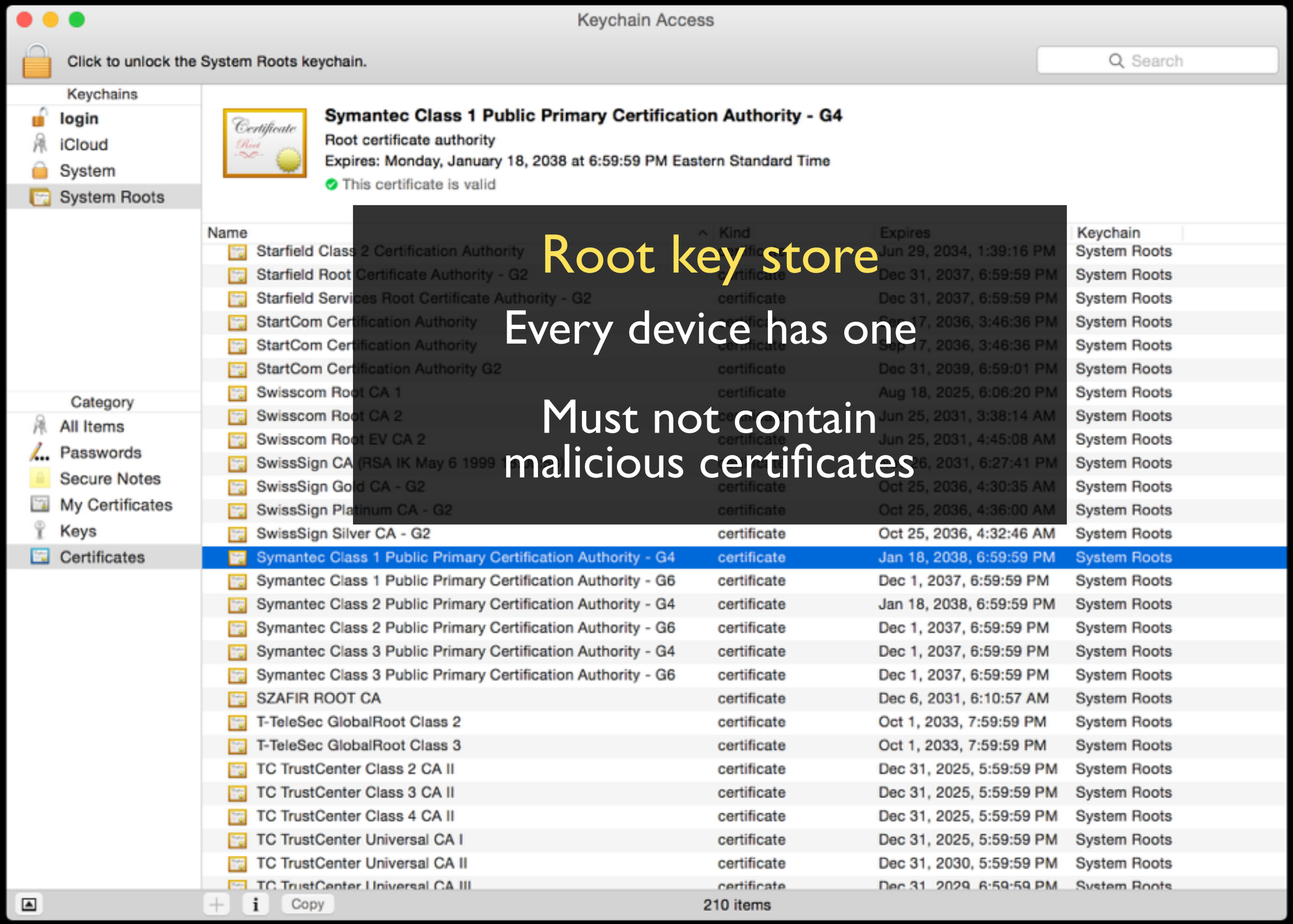
"I'm because says so"

"I'm because says so"

Verifying certificates



Verifying certificates



Verifying certificates

"I'm because I say so!"

"I'm because says so"

"I'm because says so"

Verifying certificates

“I’m because I say so!”

“I’m because says so”

“I’m because says so”

Verifying certificates

"I'm because I say so!"

"I'm because says so"

"I'm because says so"

Verifying certificates

"I'm because I say so!"

"I'm because says so"

"I'm because says so"

Verifying certificates

"I'm because I say so!"

"I'm because says so"

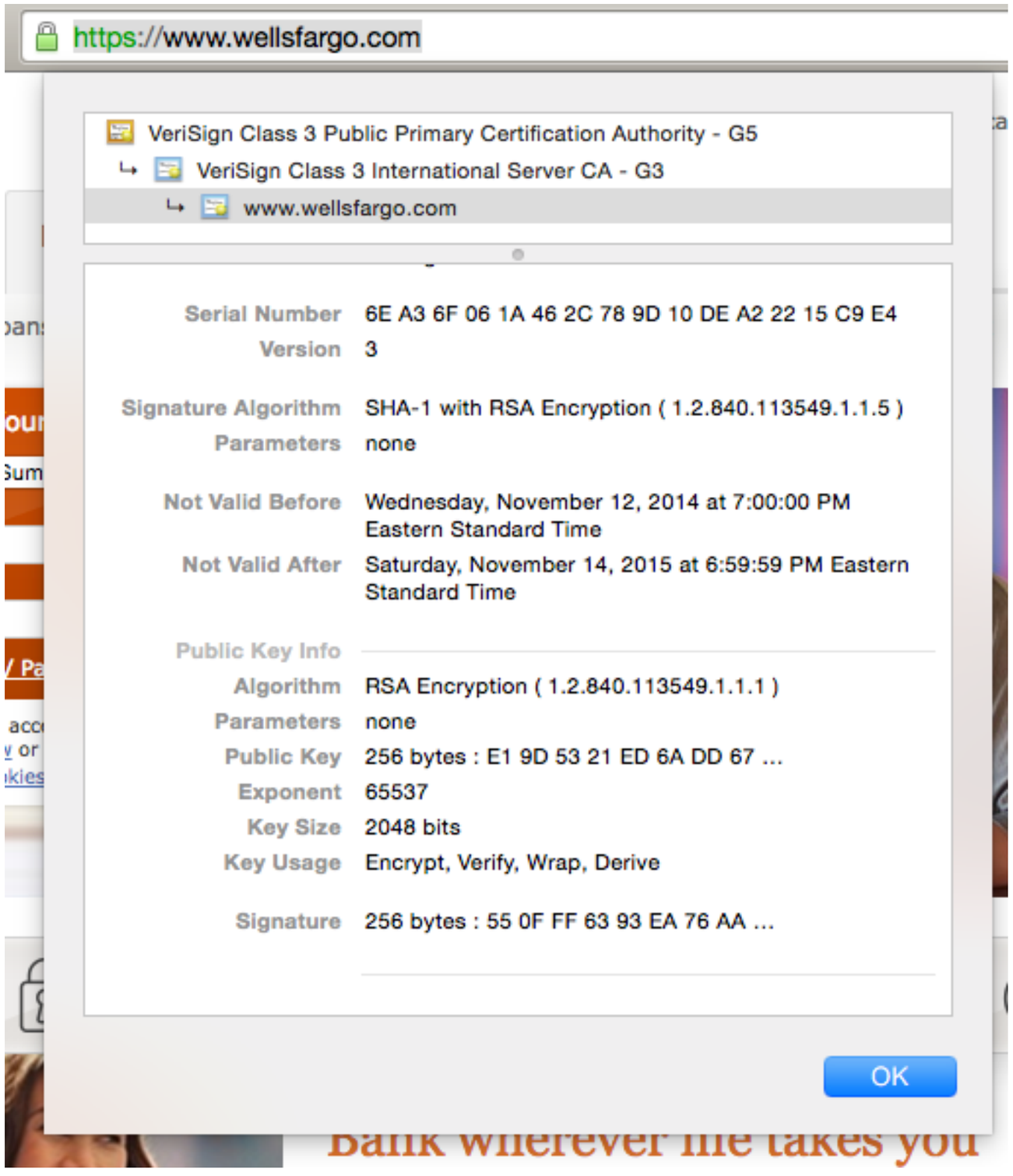
"I'm because says so"

Verifying certificates

"I'm because I say so!"

"I'm because says so"

"I'm because says so"

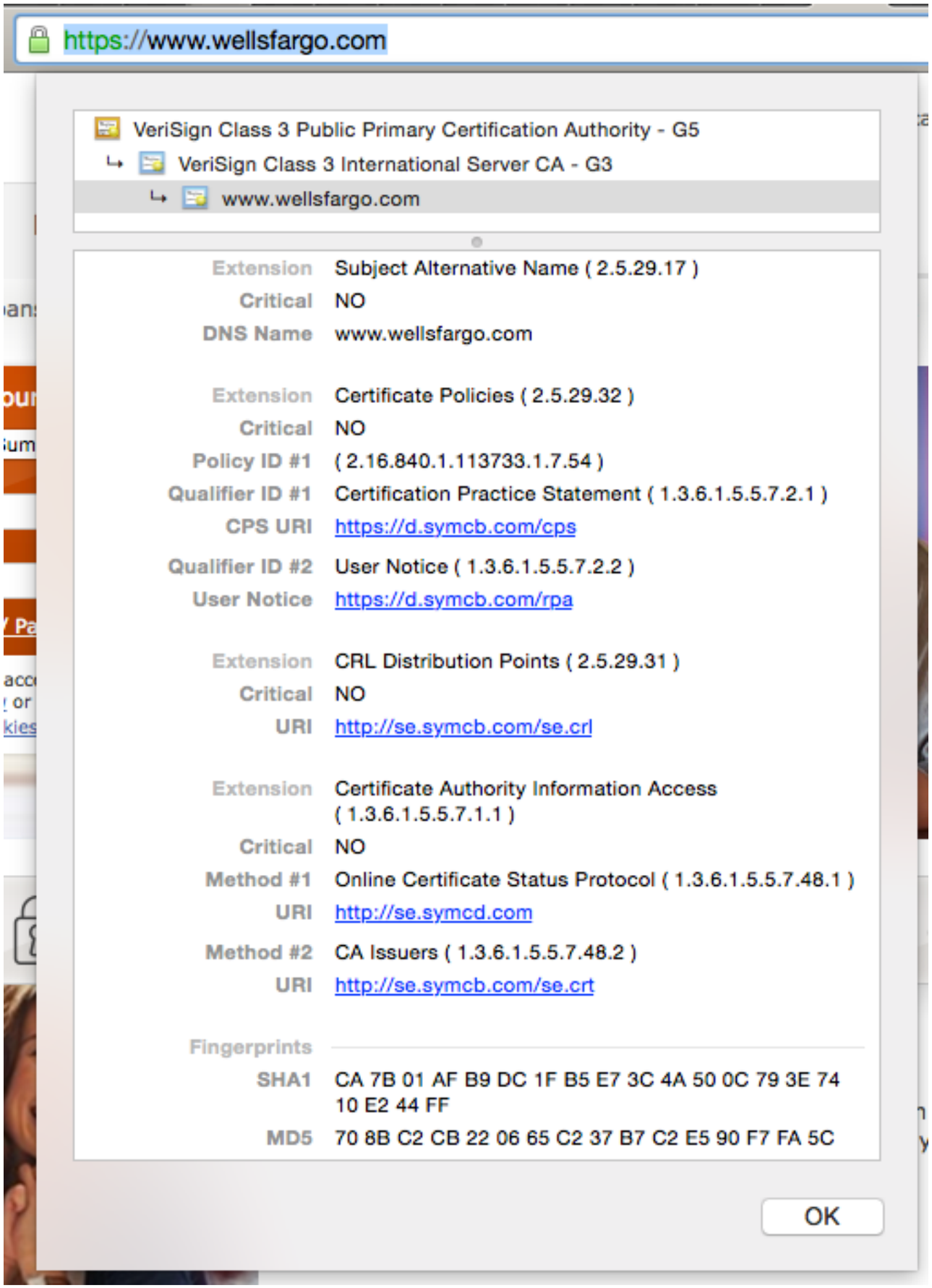


Serial number: Uniquely identifies this cert with respect to the issuer (look for this in CRLs)

Signature algorithm: How the issuer will sign parts of the cert

Not valid before/after: When to start and stop believing this cert (start & expiration dates)

The public key: And the issuer's signature of the public key



Subject Alternate Names:
Other URLs for which this cert should be considered valid. (wellsfargo.com is not the same as www.wellsfargo.com)

Can include wildcards, e.g., *.google.com

CRL & OCSP:
Where to go to check if this certificate has been revoked

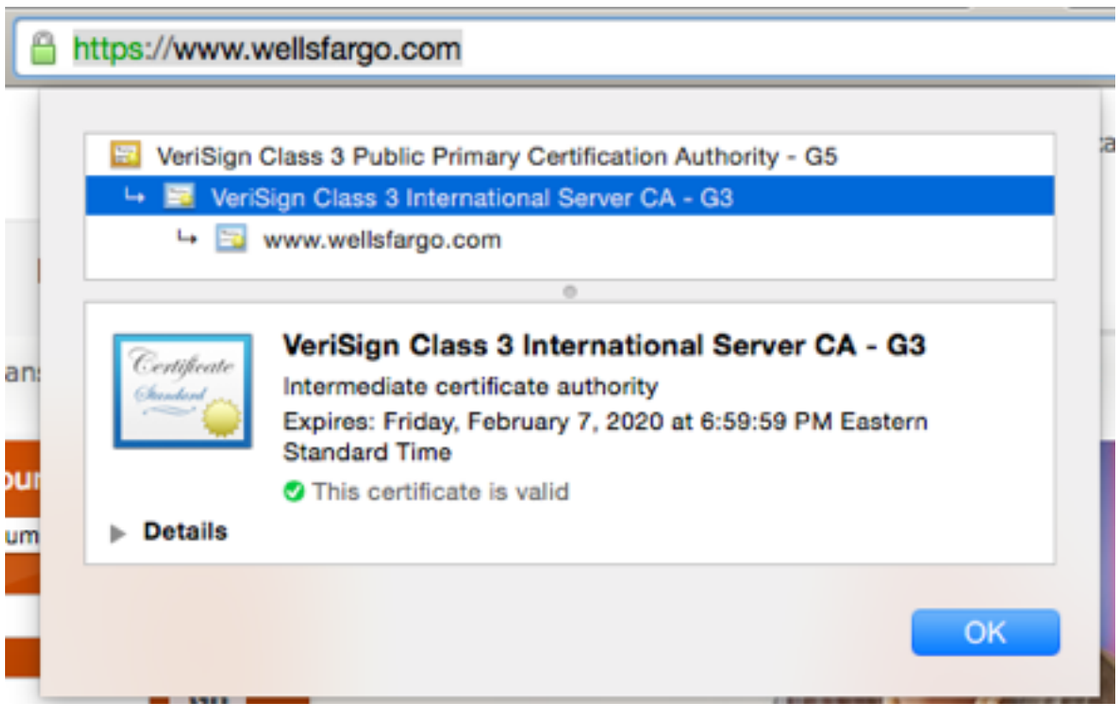
Non-cryptographic checksums

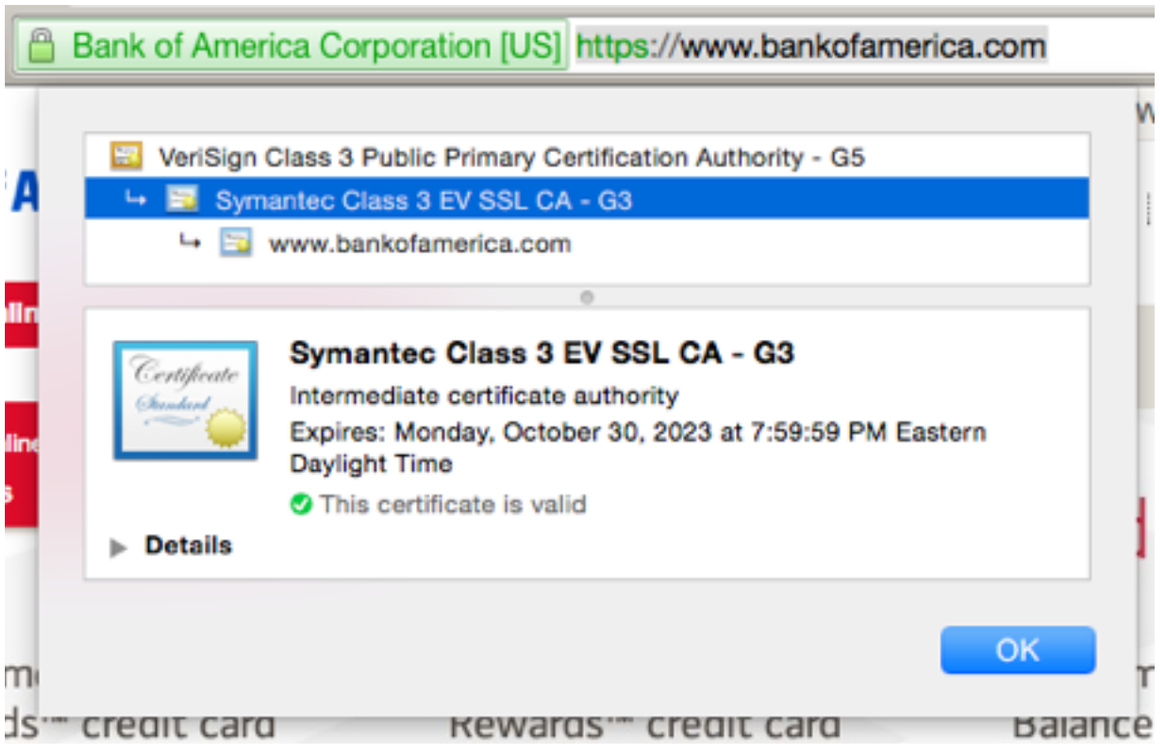
Certificate types

Why are these different?

Certificate types

Why are these different?





This is an EV (extended validation) certificate; browsers show the full name for these kinds of certs

Proper reaction to Heartbleed

1. Patch the software

2. “Reissue” a new key (get a new one and load it onto your servers)

3. Revoke the old key

Proper reaction to Heartbleed

1. Patch the software

2. “Reissue” a new key (get a new one and load it onto your servers)

3. Revoke the old key

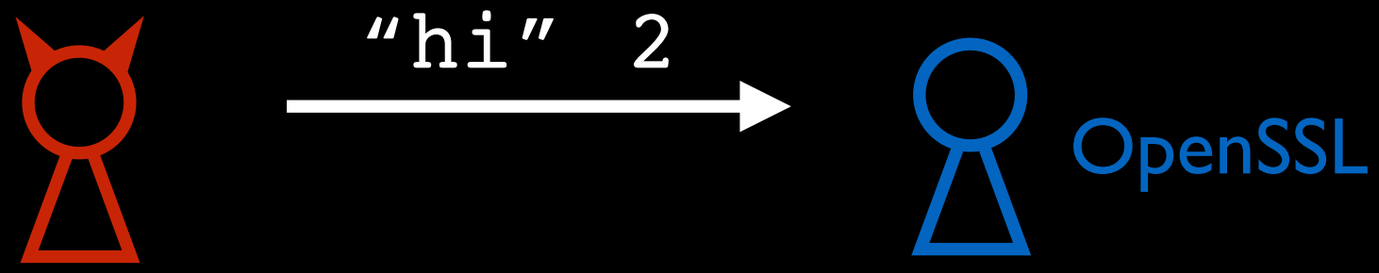
<u>Order matters!</u>
If we reissued and then patched,
then our new key would be compromised, too.

If we revoked first, we’d be offline.

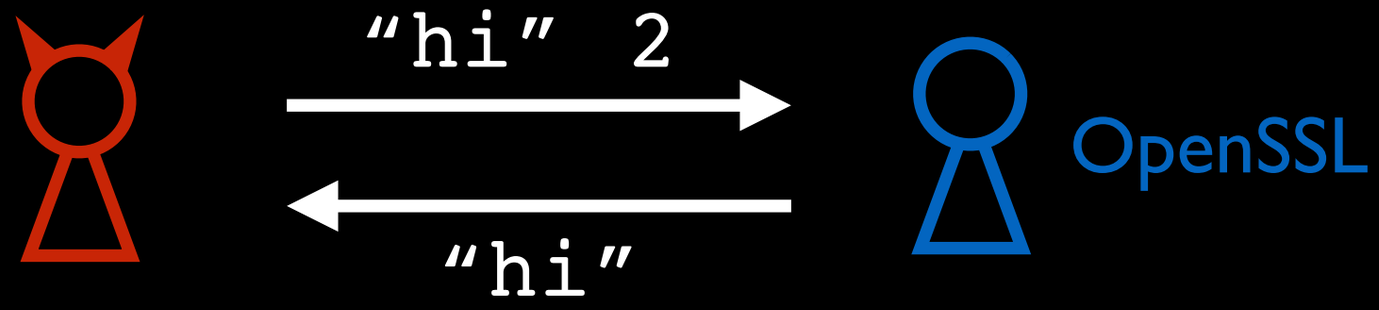
Heartbleed

OpenSSL

Heartbleed



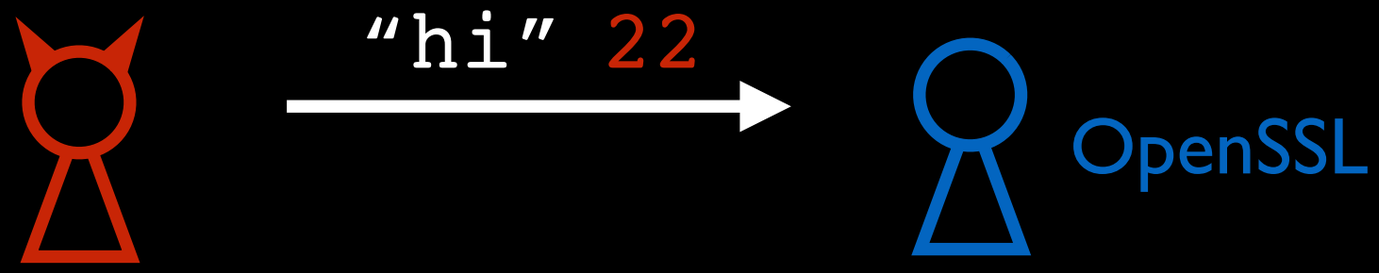
Heartbleed



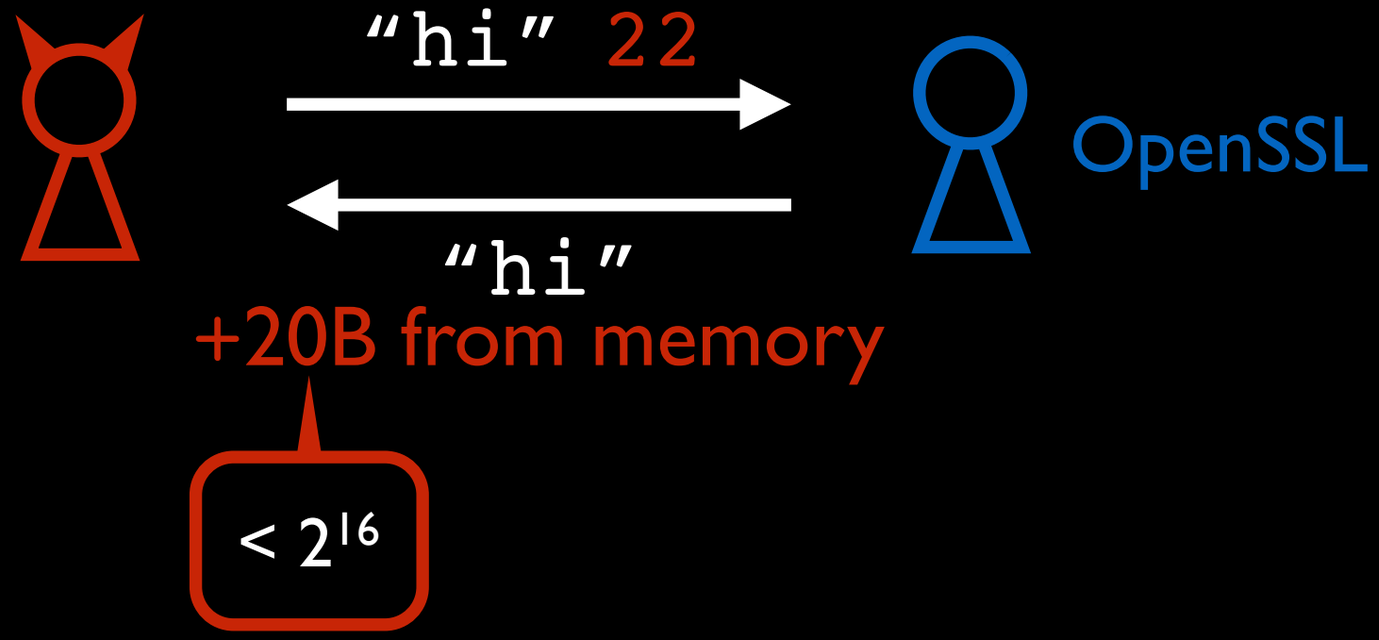
Heartbleed

OpenSSL

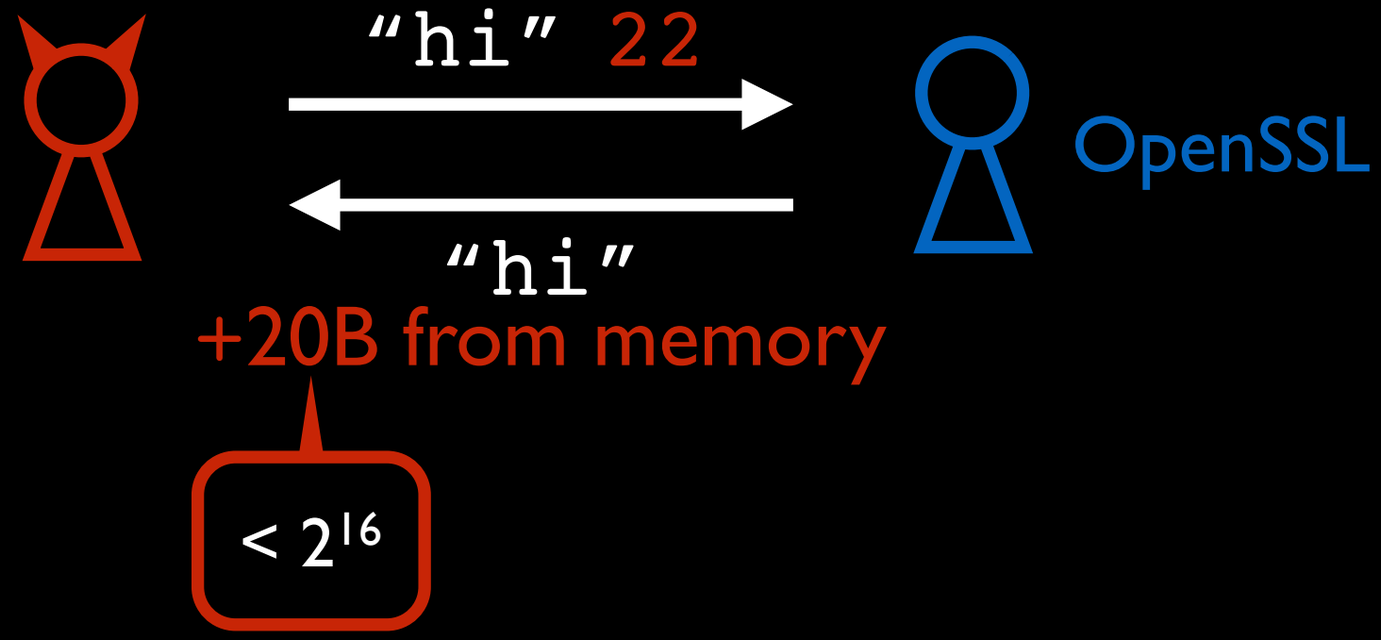
Heartbleed



Heartbleed



Heartbleed

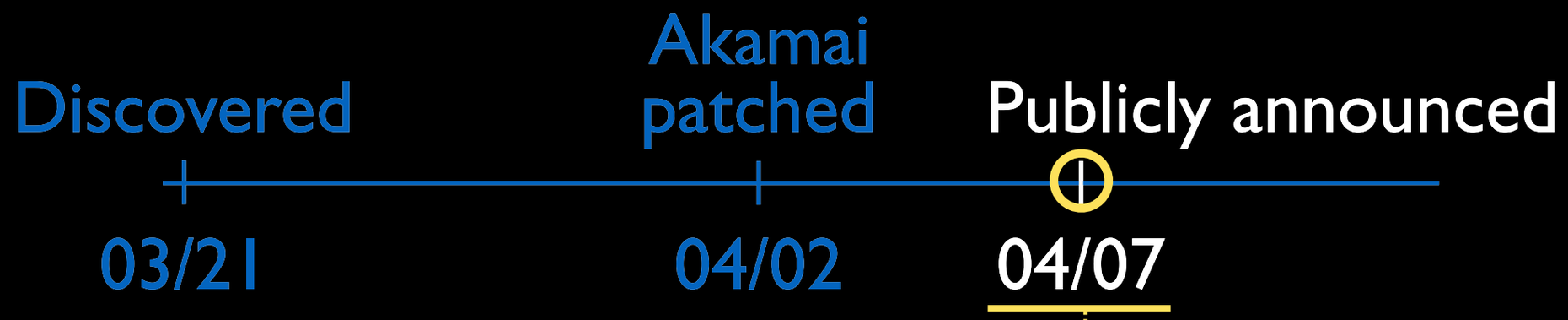


Potentially reveals user data and private keys

Heartbleed exploits were undetectable

Why study Heartbleed?

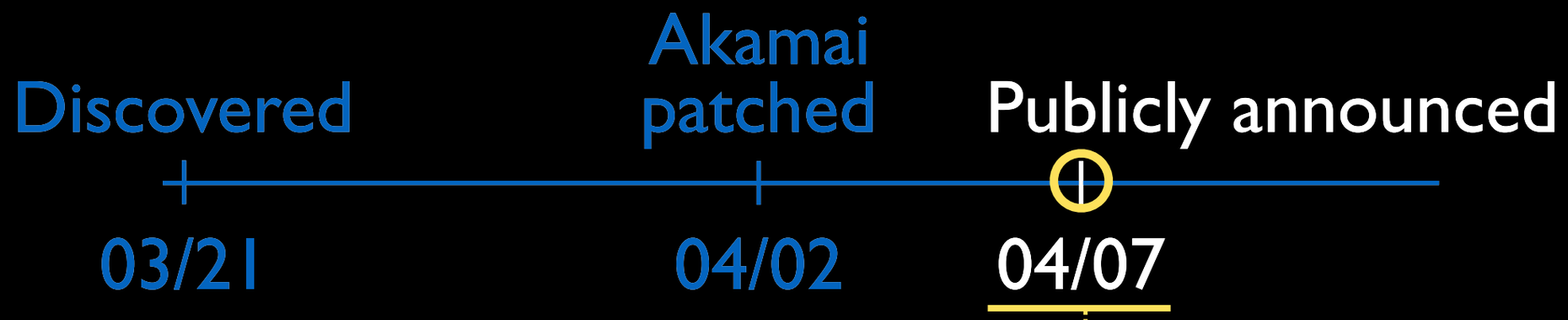
Why study Heartbleed?



Every vulnerable website should have:

1. Patched
2. Revoked
3. Reissued

Why study Heartbleed?



Every vulnerable website should have:

1. Patched
2. Revoked
3. Reissued

Heartbleed is a natural experiment:

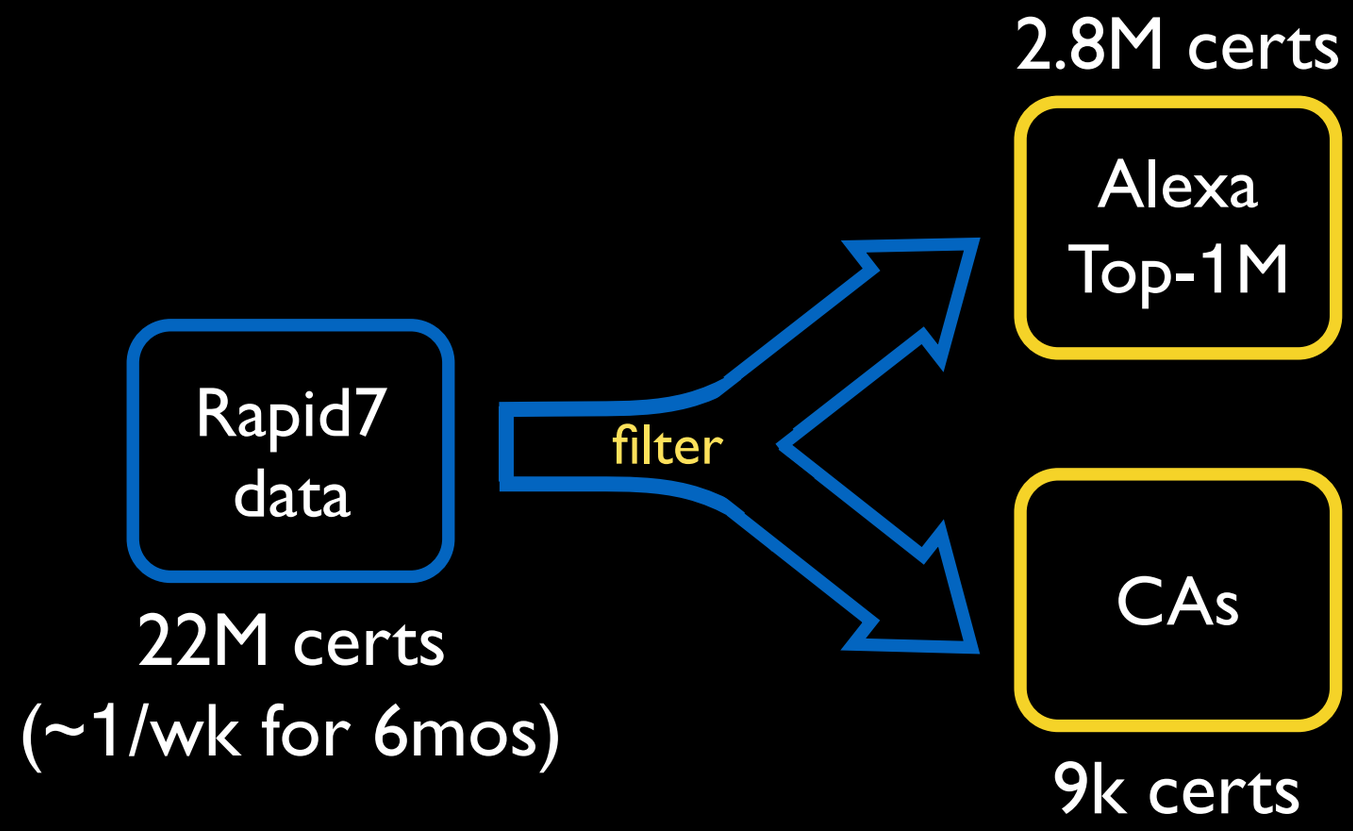
How quickly and thoroughly do administrators act?

Dataset

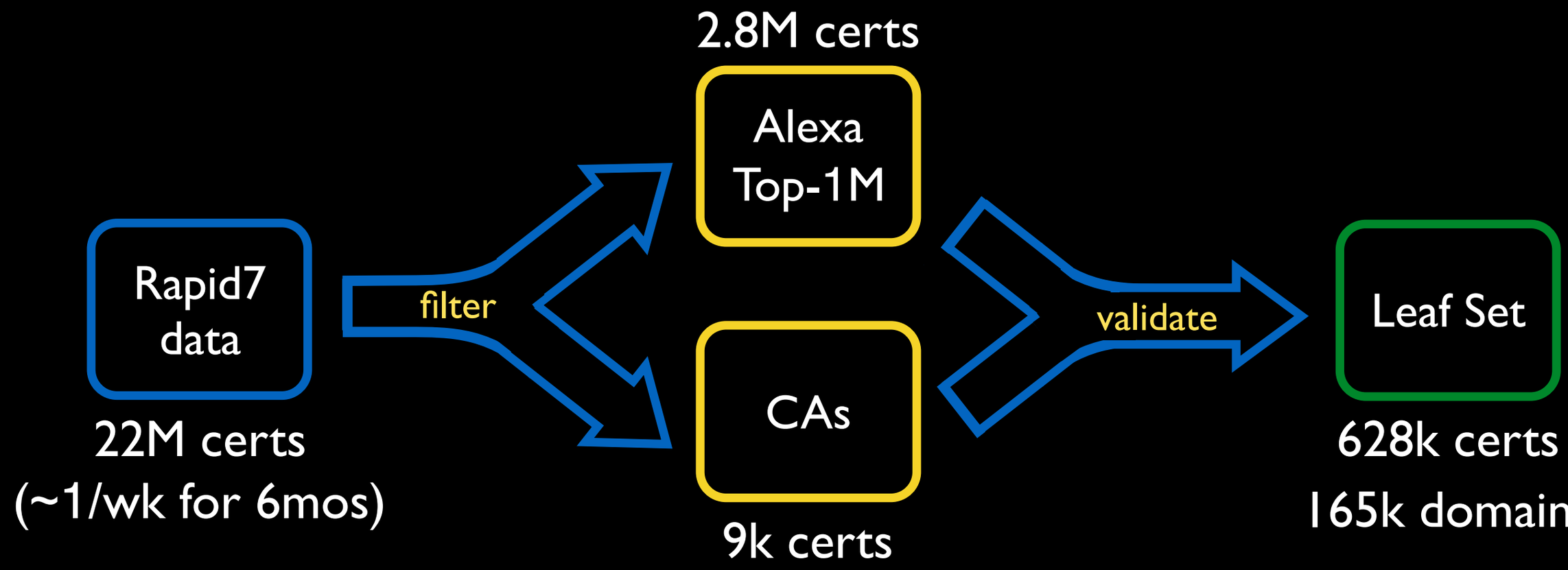
Rapid7
data

22M certs
(~1/wk for 6mos)

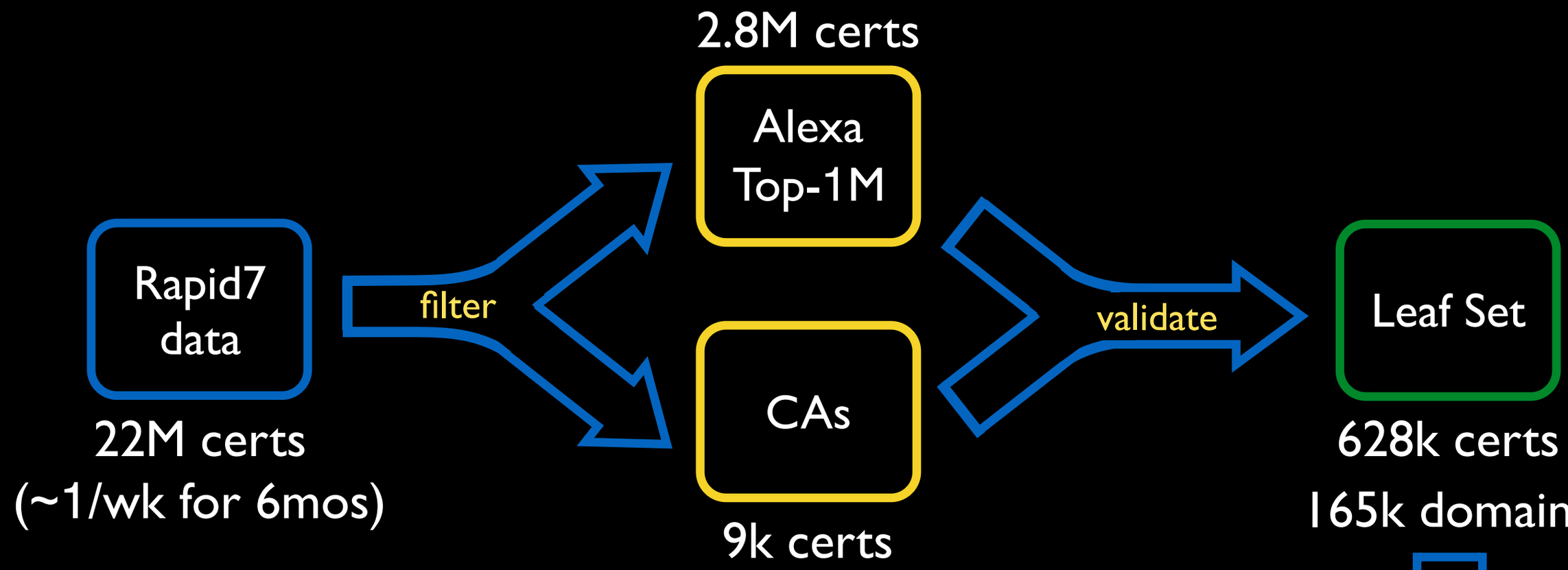
Dataset


2.8M certs
Alexa
Top-1M
Rapid7
data
filter
22M certs
(~1/wk for 6mos)
CAs
9k certs

Dataset

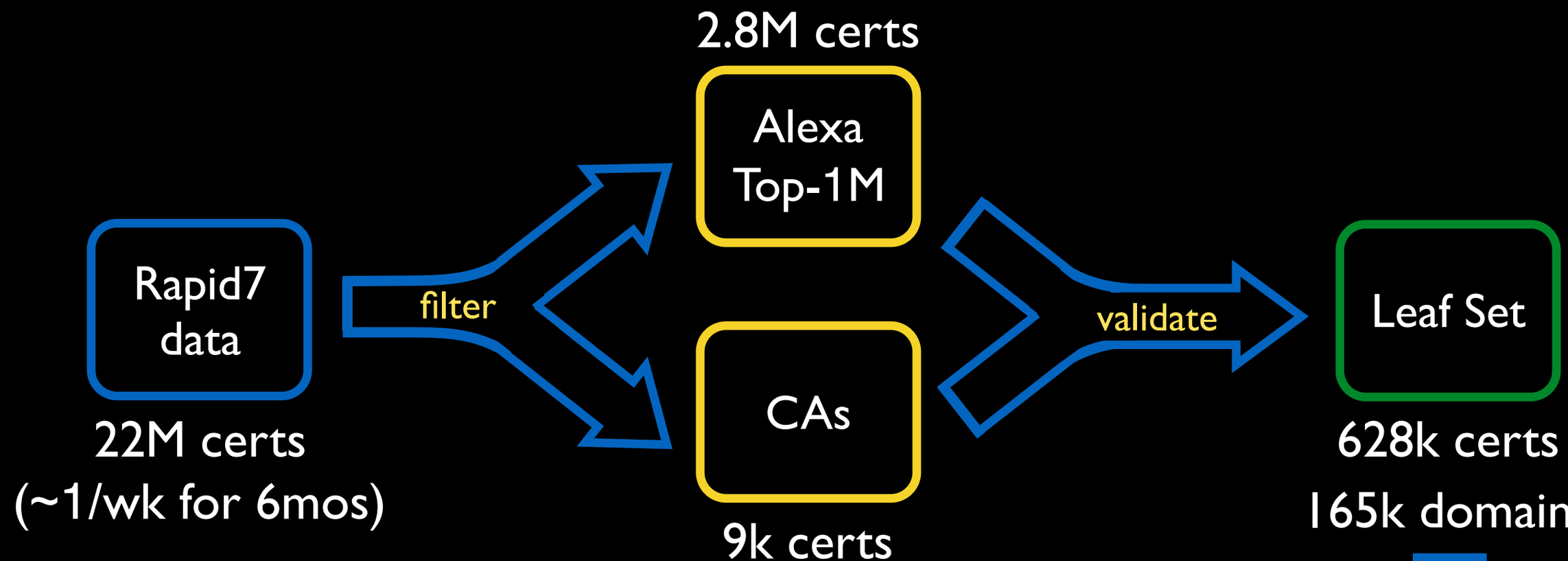

2.8M certs
Alexa Top-1M
Rapid7 data
filter
validate
Leaf Set
CAs
22M certs
(~1/wk for 6mos)
9k certs
628k certs
165k domains

Dataset



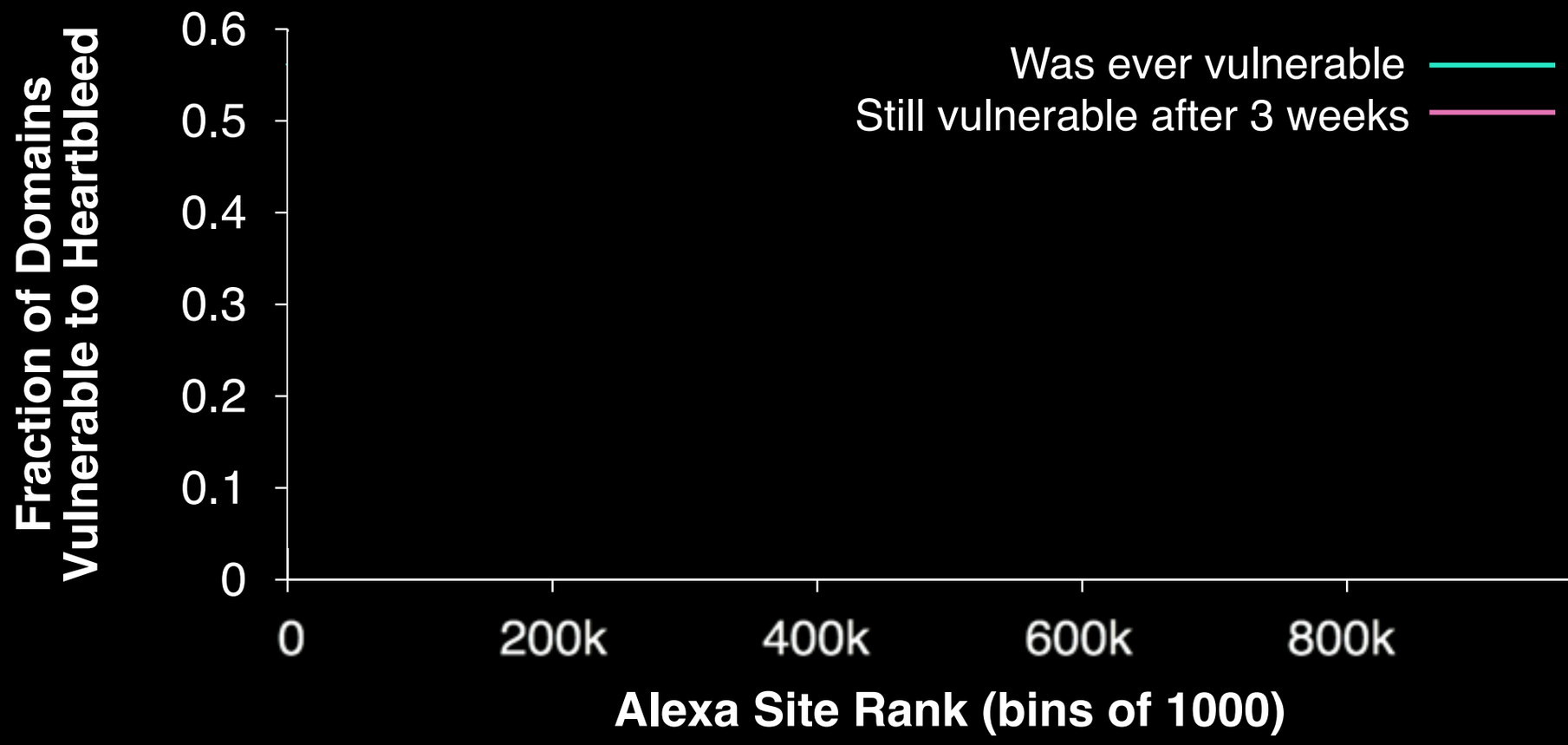
- Download CRLs
- Detect vulnerability
- Identify *Heartbleed-induced* reissues & revocations

Dataset

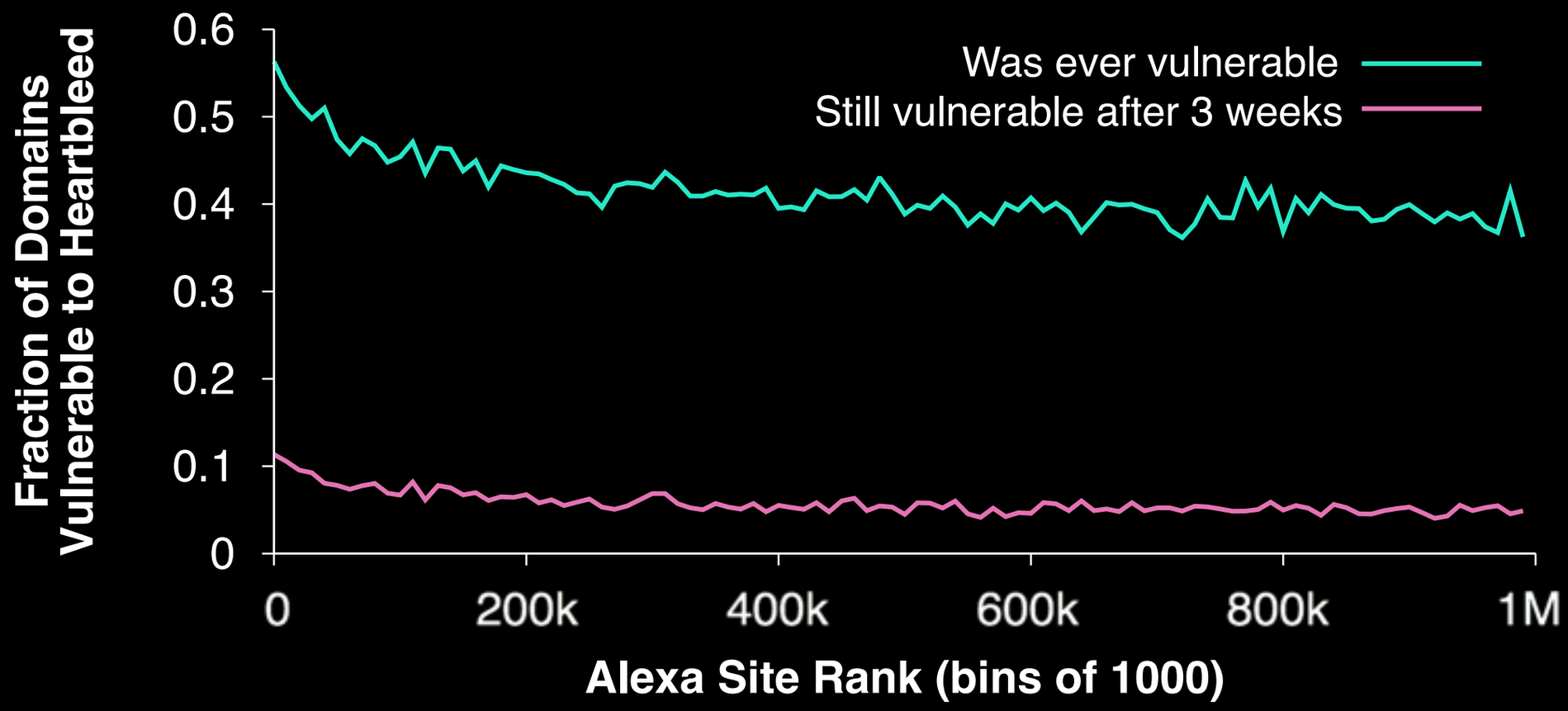


- Download CRLs
- Detect vulnerability
- Identify *Heartbleed-induced* reissues & revocations

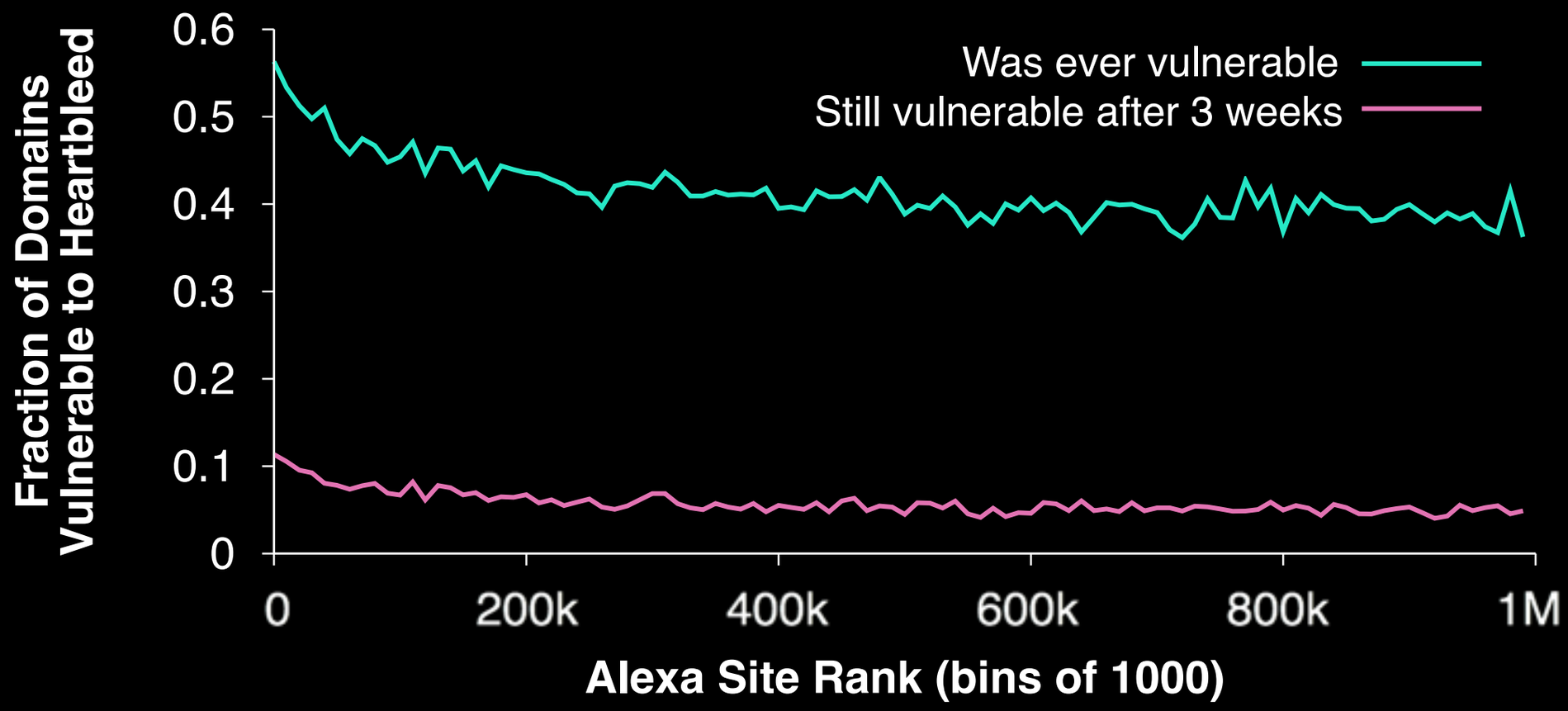
Prevalence and patch rates



Prevalence and patch rates



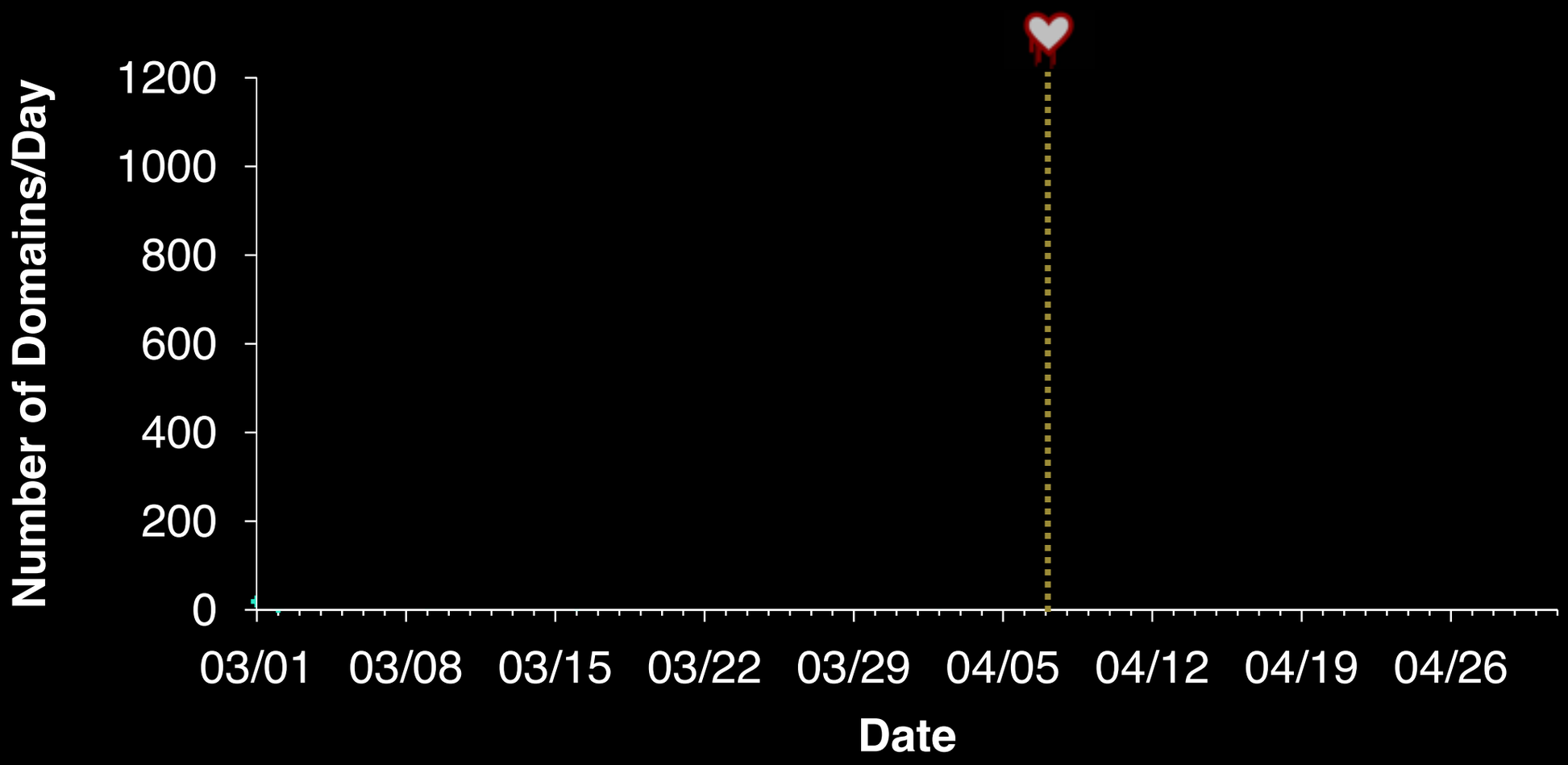
Prevalence and patch rates



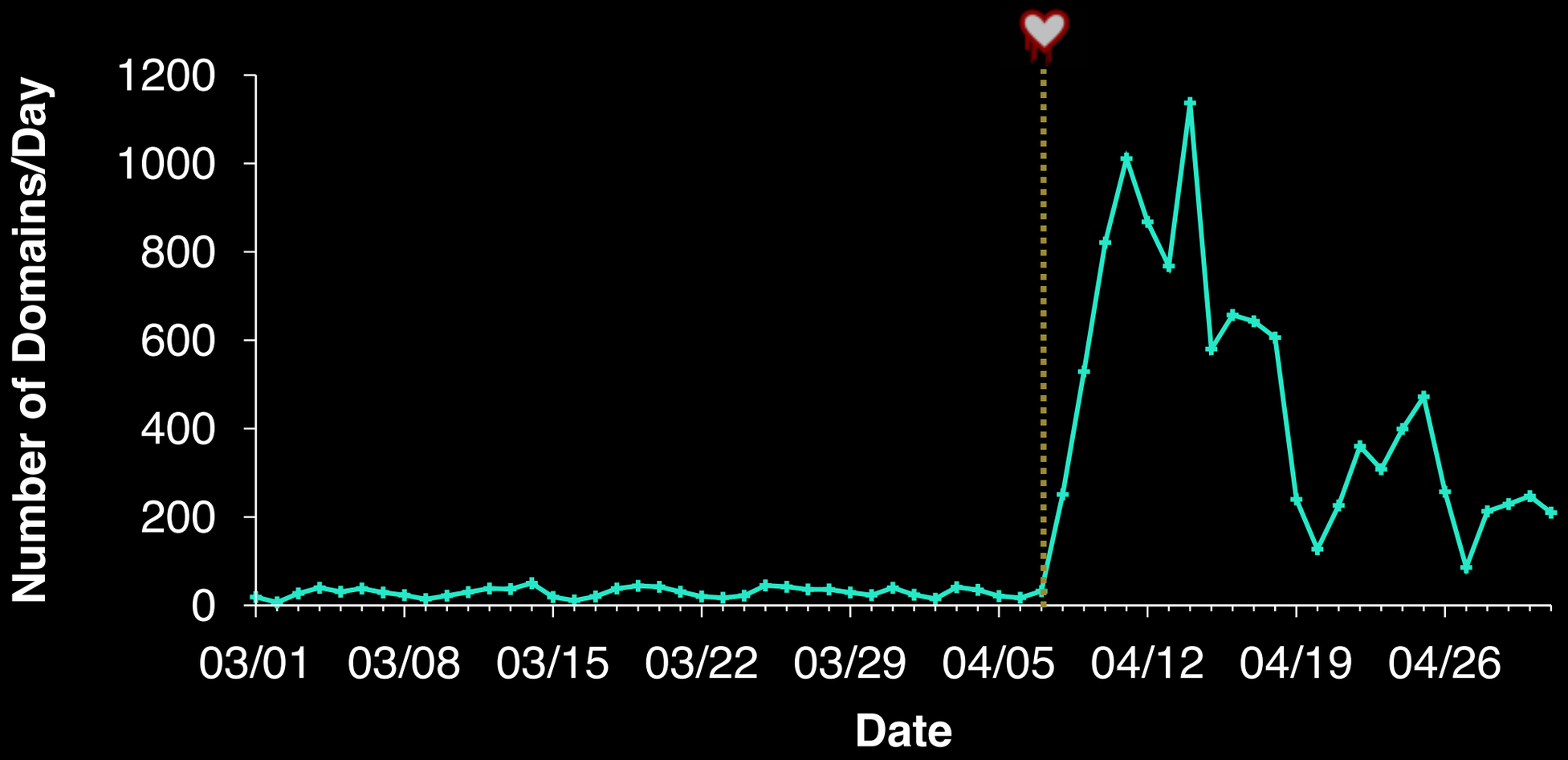
Patching rates are mostly positive

Only ~7% had not patched within 3 weeks

How quickly were certs revoked?

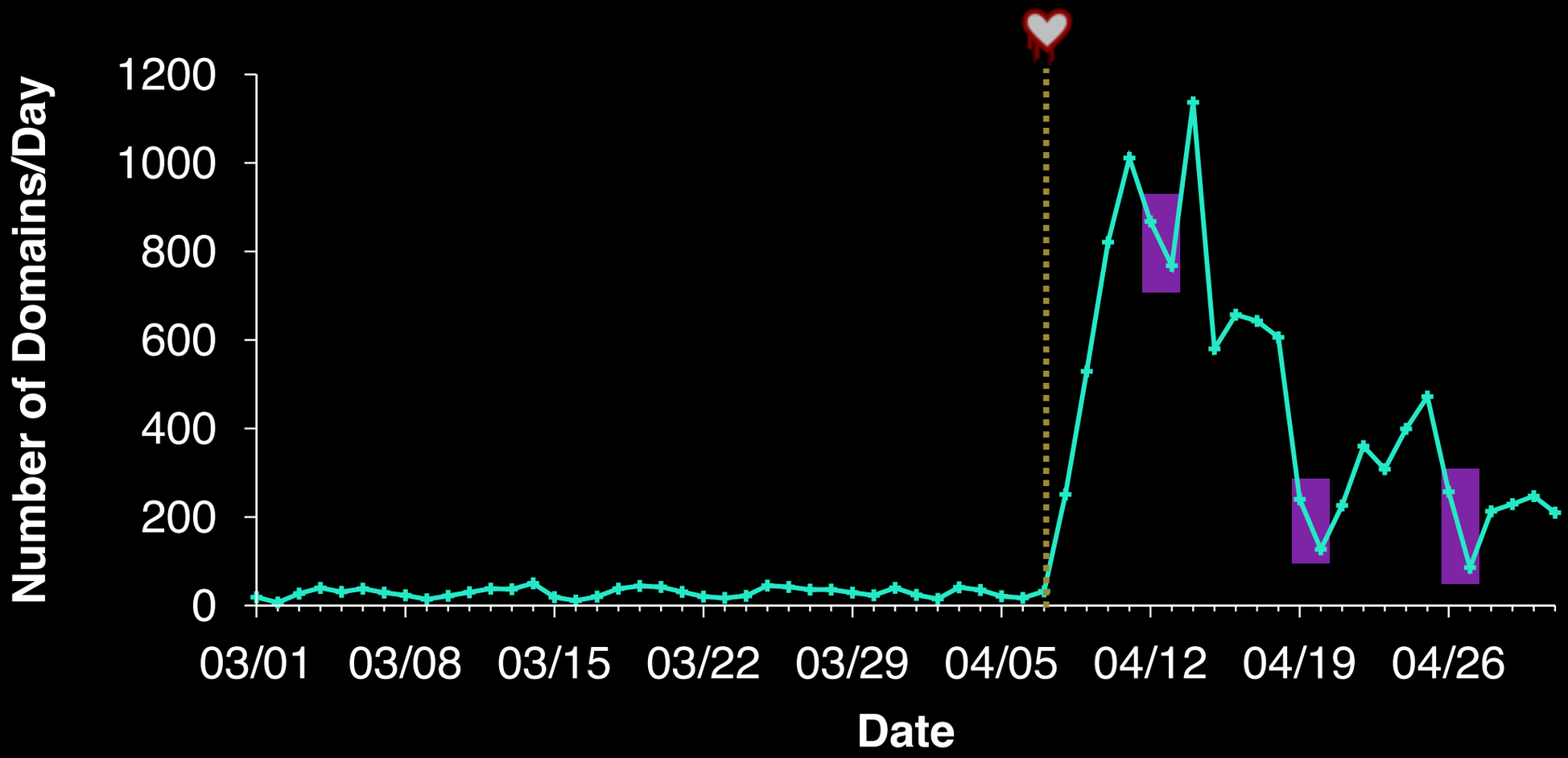


How quickly were certs revoked?



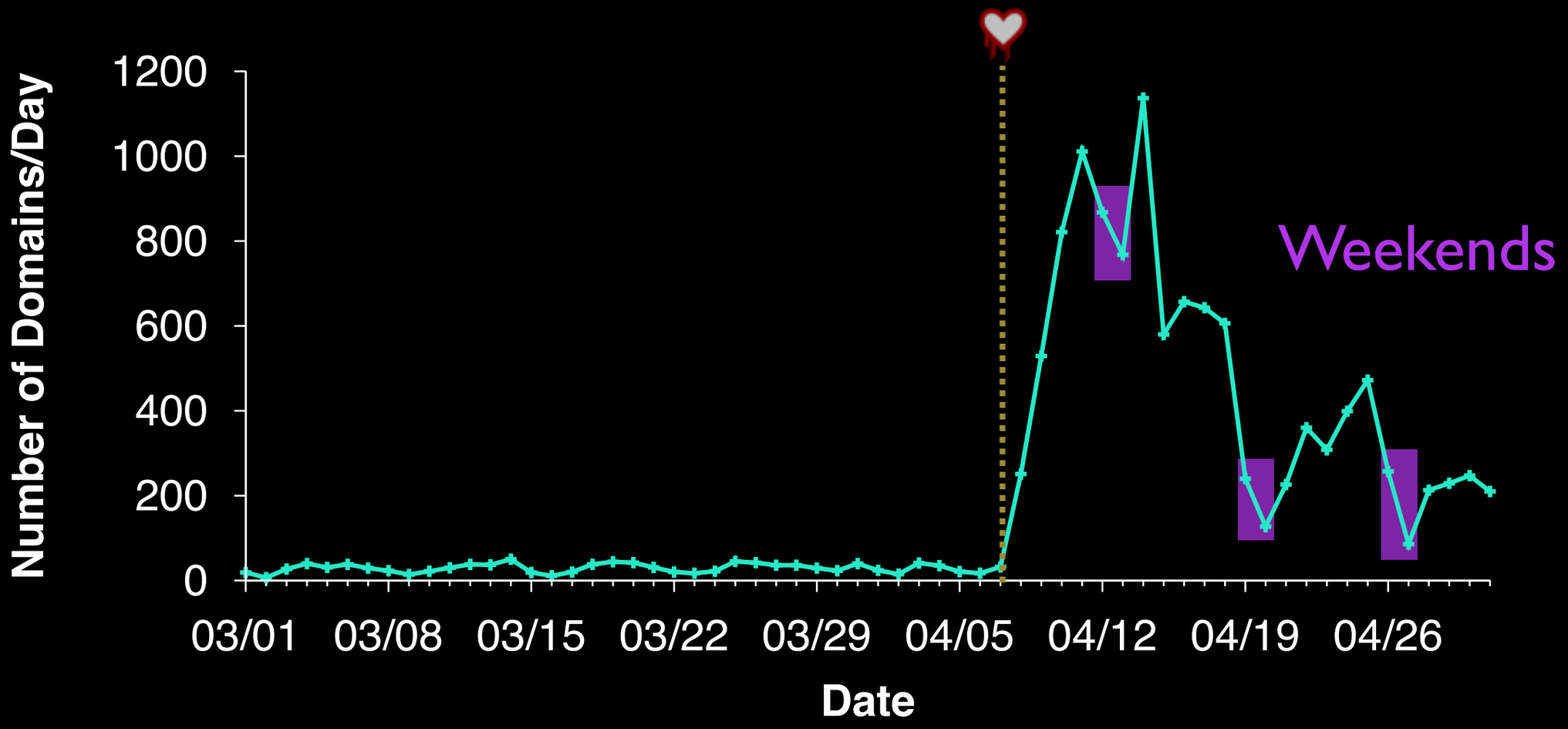
Reaction ramps up quickly

How quickly were certs revoked?



Reaction ramps up quickly

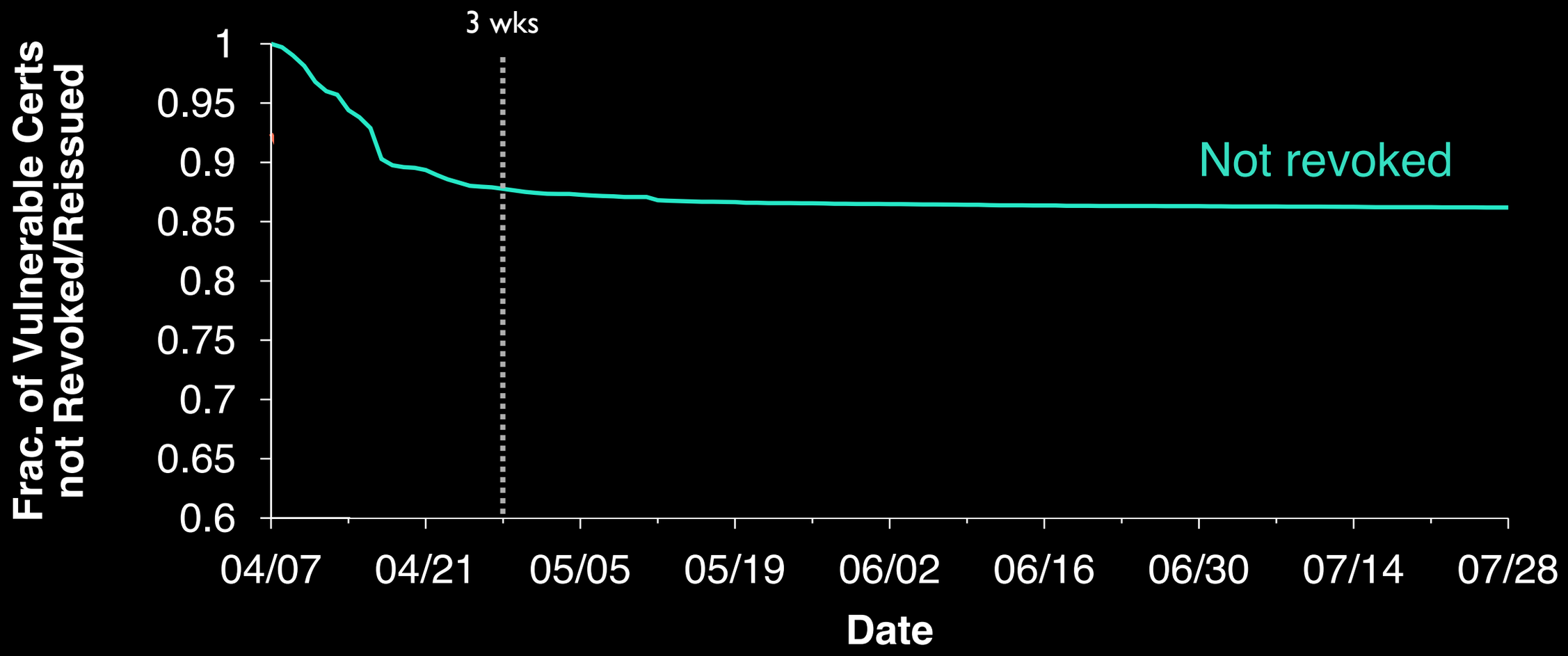
How quickly were certs revoked?



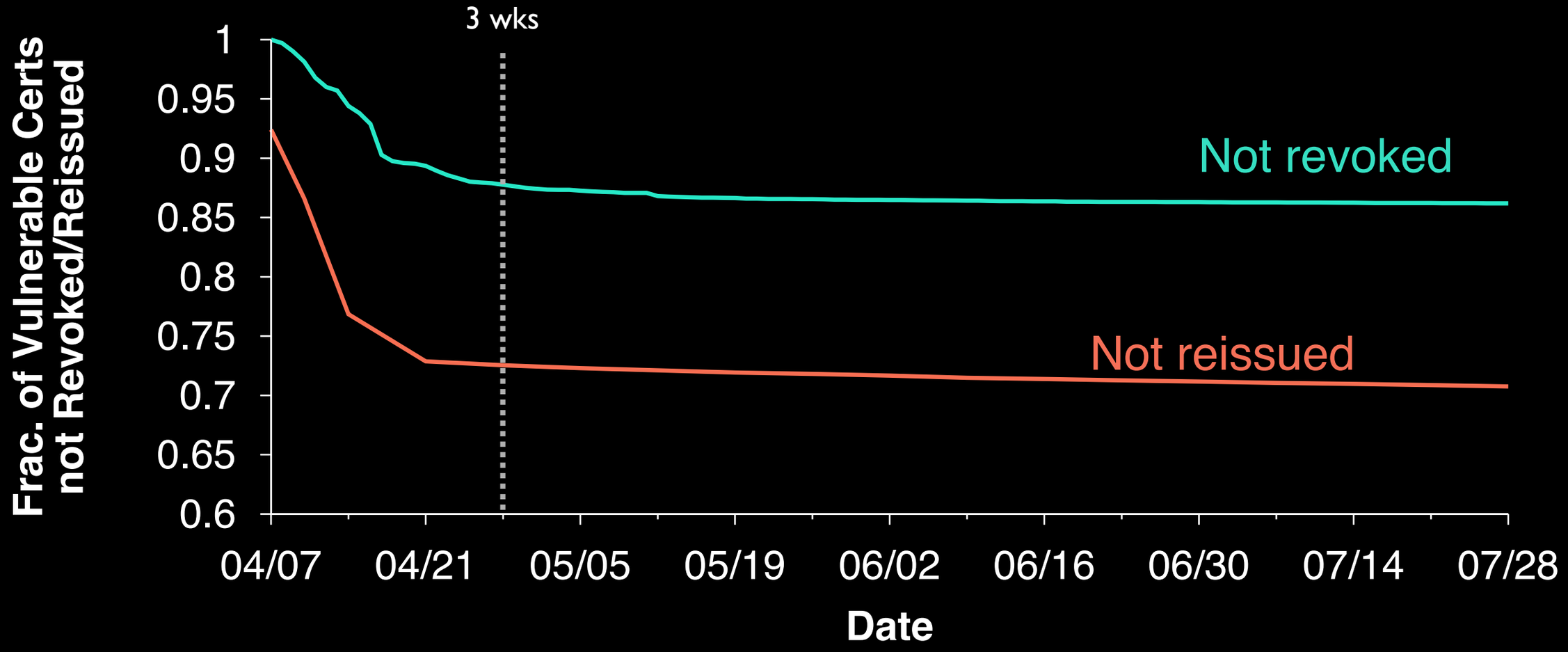
Reaction ramps up quickly

Security takes the weekends off

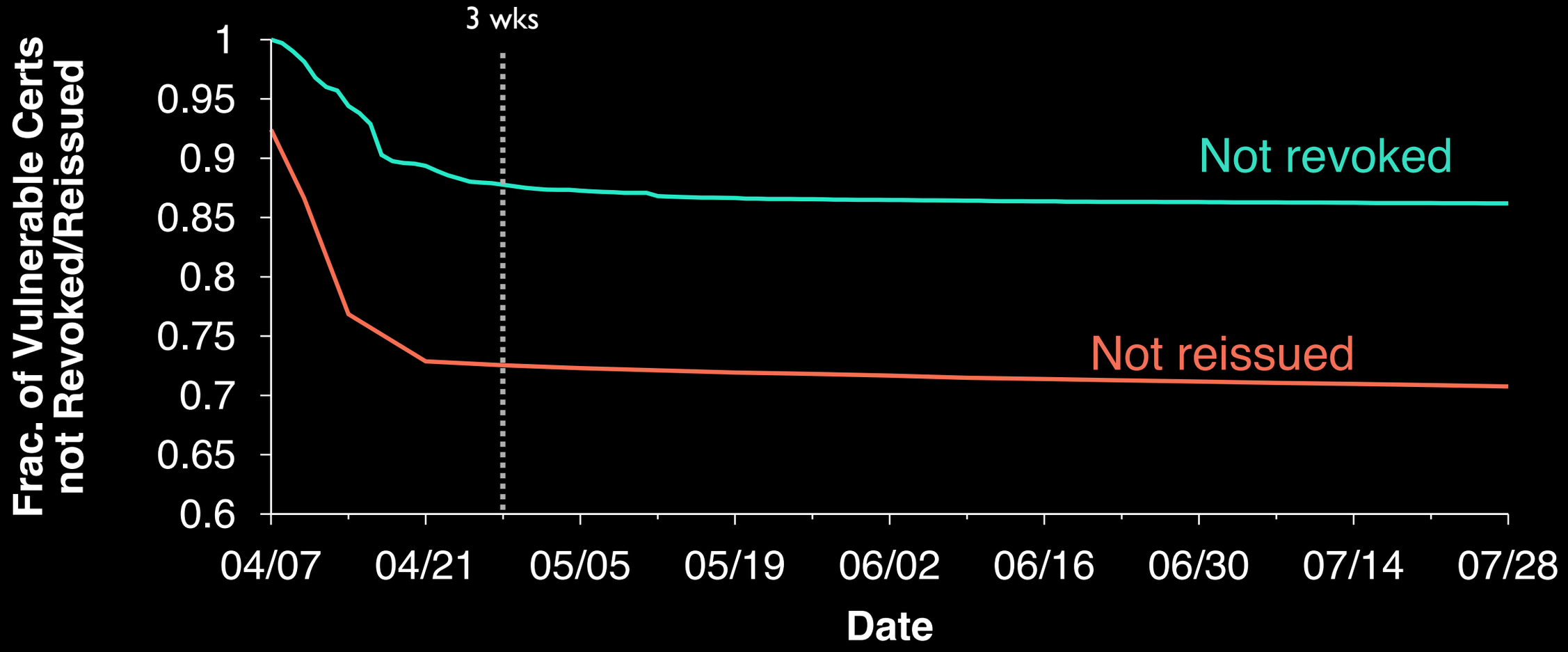
Certificate update rates



Certificate update rates



Certificate update rates



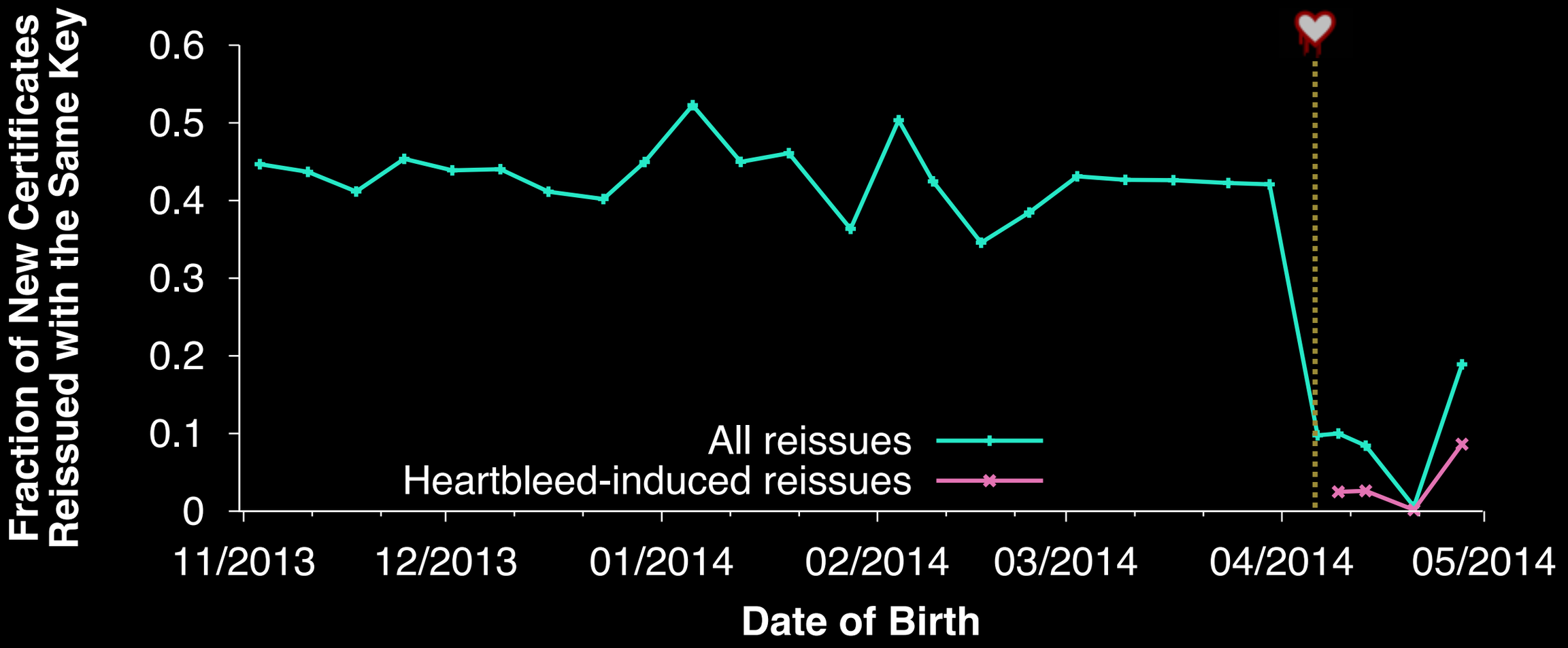
Similar pattern to patches:

Exponential drop-off, then levels out

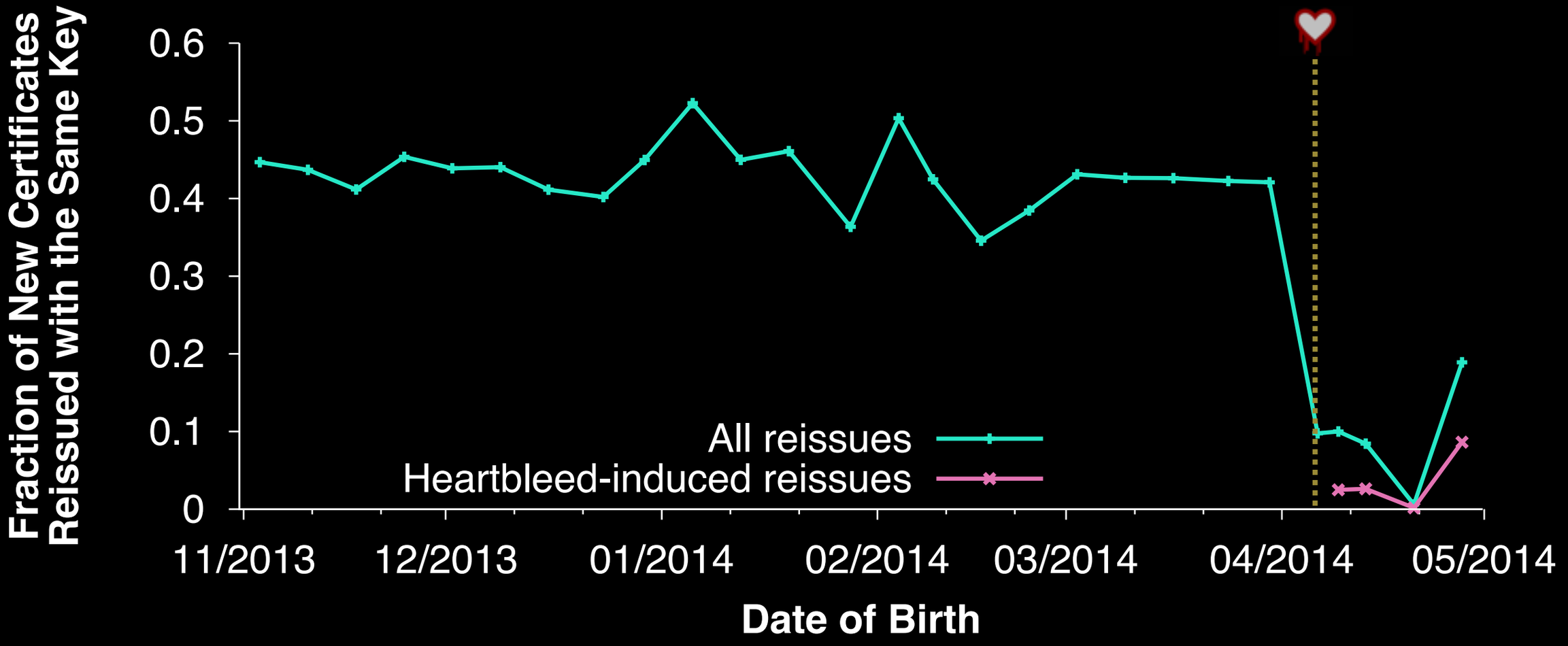
After 3 weeks: 13% Revoked 27% Reissued

Reissue ⇒ New key?

Reissue ⇒ New key?



Reissue ⇒ New key?



Reissuing the same key is common practice

4.1% Heartbleed-induced

The ugly truth of revocations

93% Patched

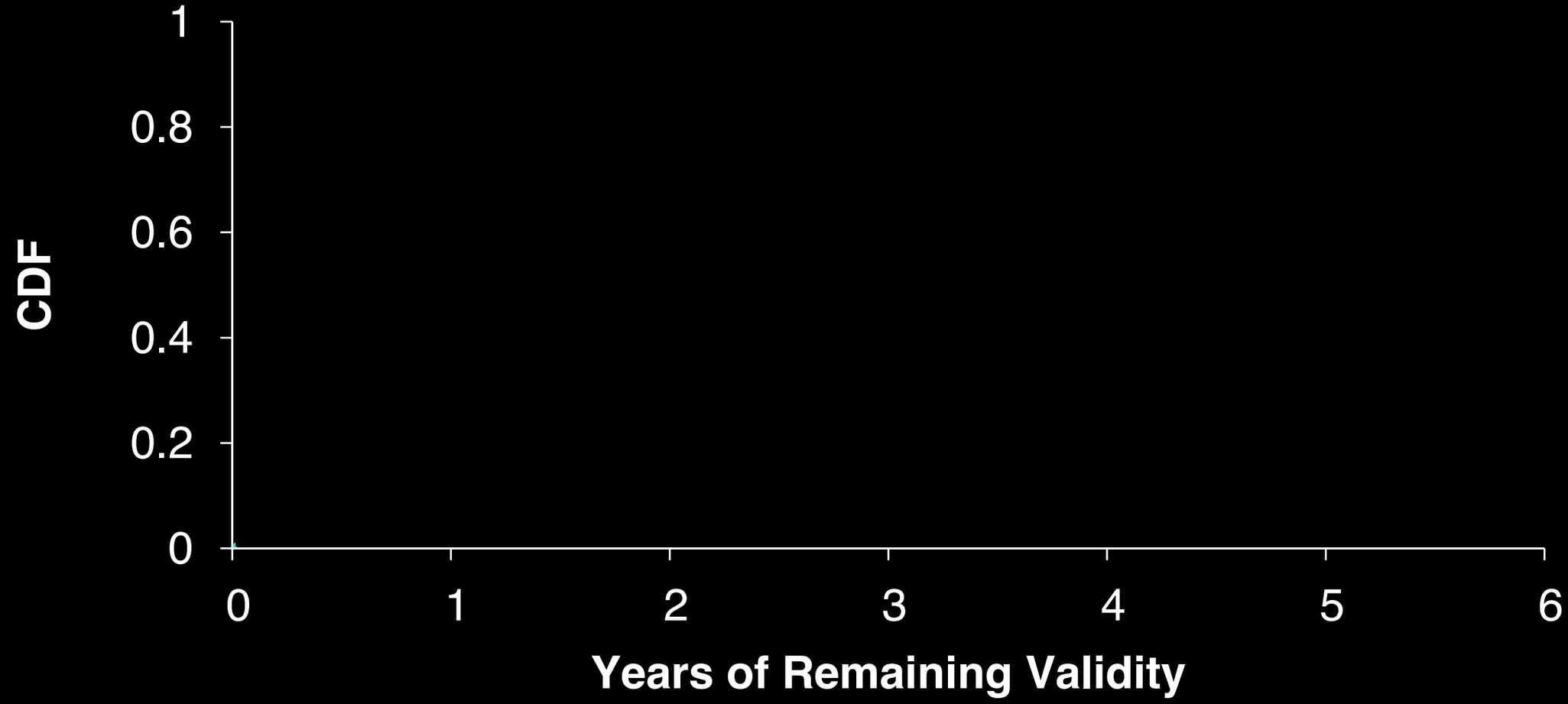
13% Revoked

27% Reissued

Security is supposed to be a fundamental design goal, but

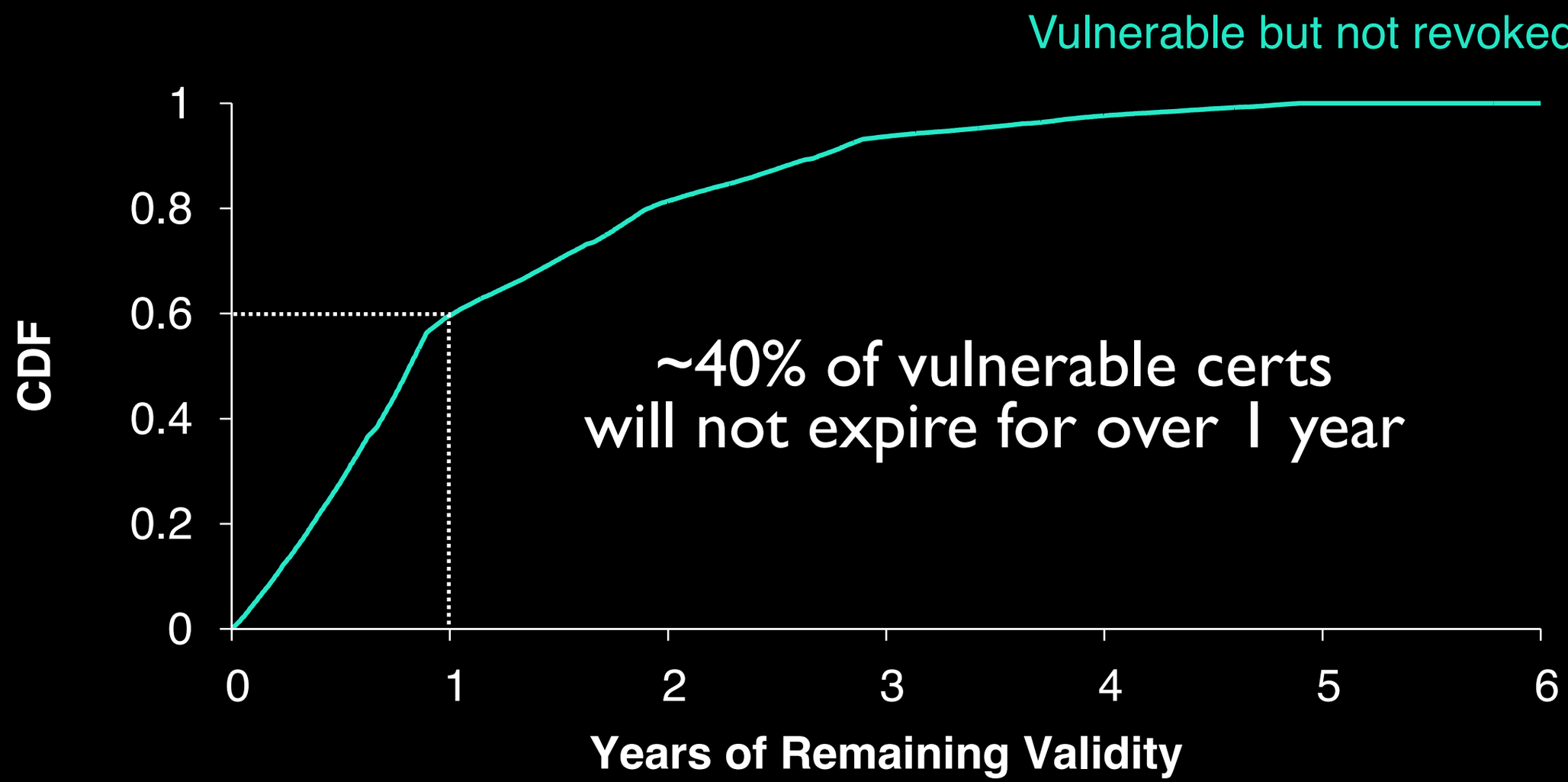
- Administrators trade off security for *ease of maintenance/cost*
- Certificate authorities trade off security for *profit*

Can we wait for expiration?

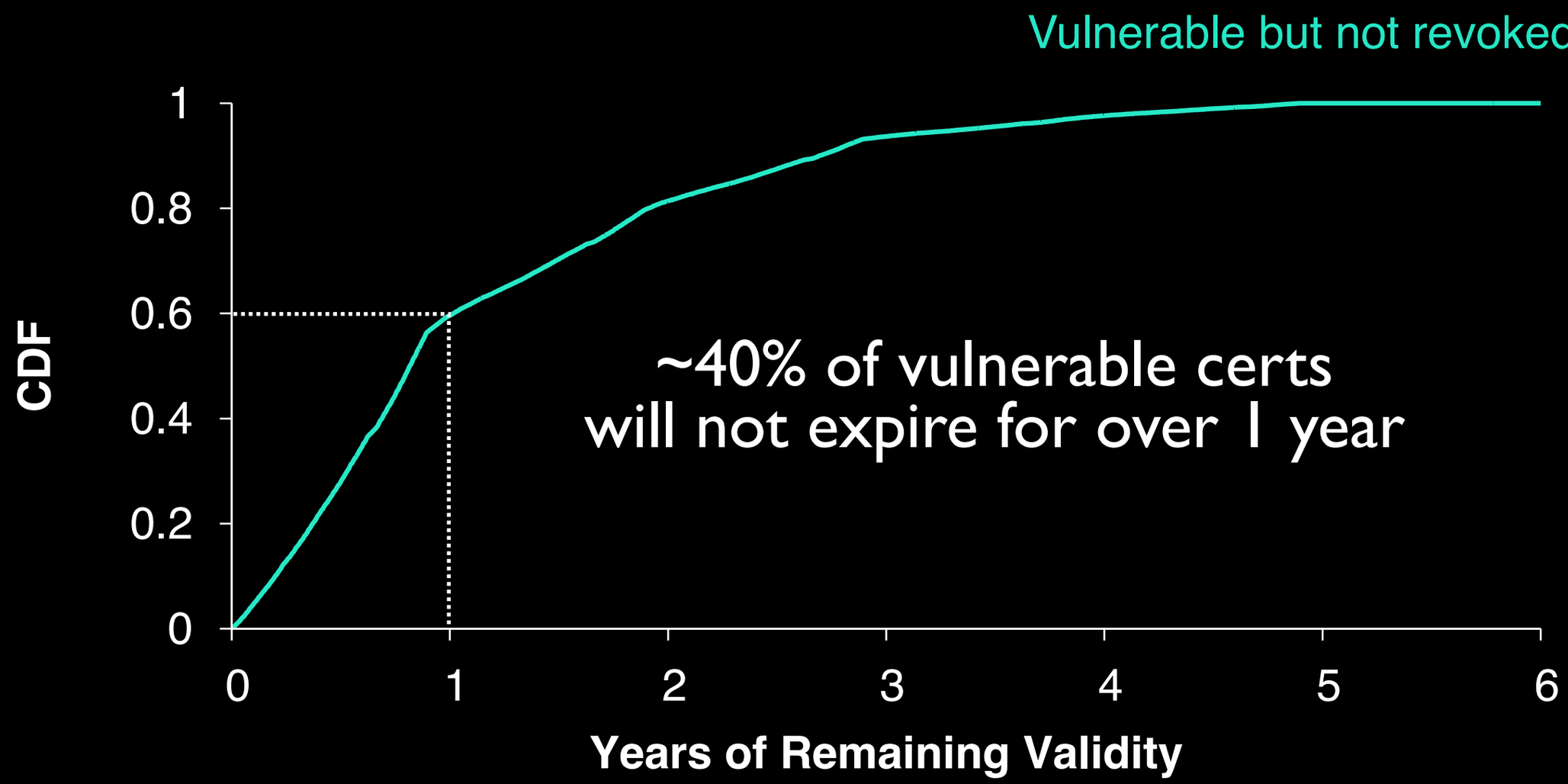


Can we wait for expiration?

Can we wait for expiration?



Can we wait for expiration?



We may be dealing with Heartbleed for *years*

Testing browser behavior

Revocation protocols

- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

Availability of revocation info

- Browsers *should* reject certs they cannot check
  - E.g., because the OCSP server is down

Chain lengths

- Browsers *should* reject a cert if *any* on the chain fail
  - Leaf, intermediate(s), root

Testing browser behavior

Revocation protocols

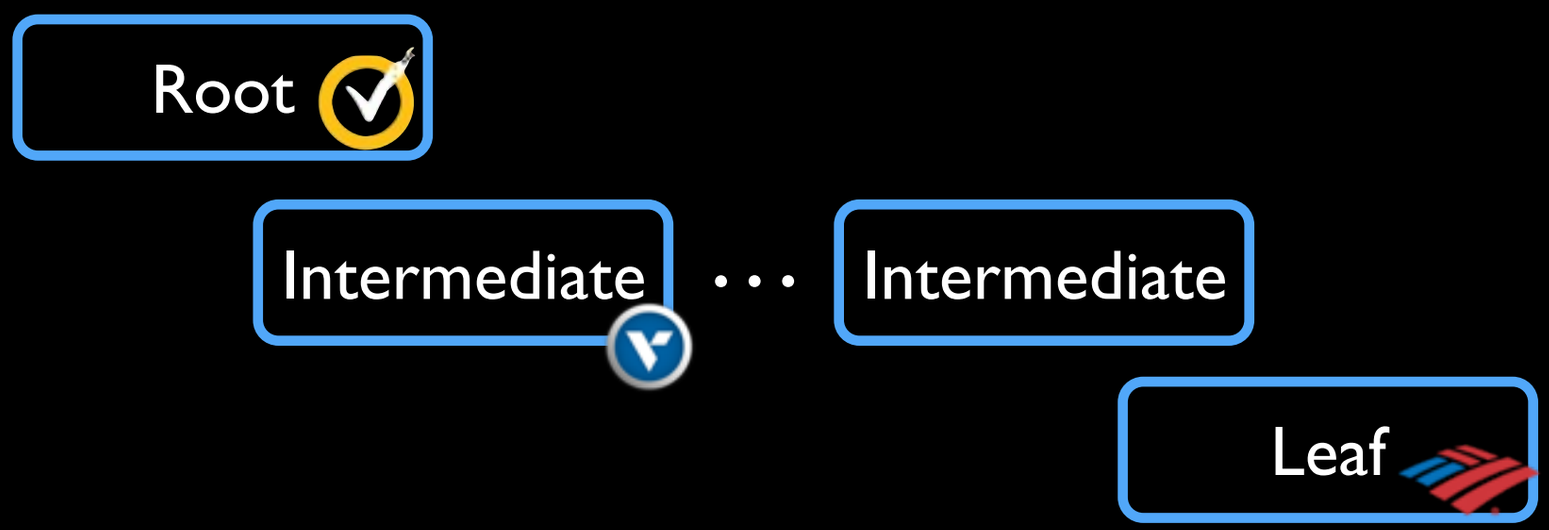
- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

Availability of revocation info

- Browsers *should* reject certs they cannot check
  - E.g., because the OCSP server is down

Chain lengths

- Browsers *should* reject a cert if *any* on the chain fail
  - Leaf, intermediate(s), root



Testing browser behavior

Revocation protocols

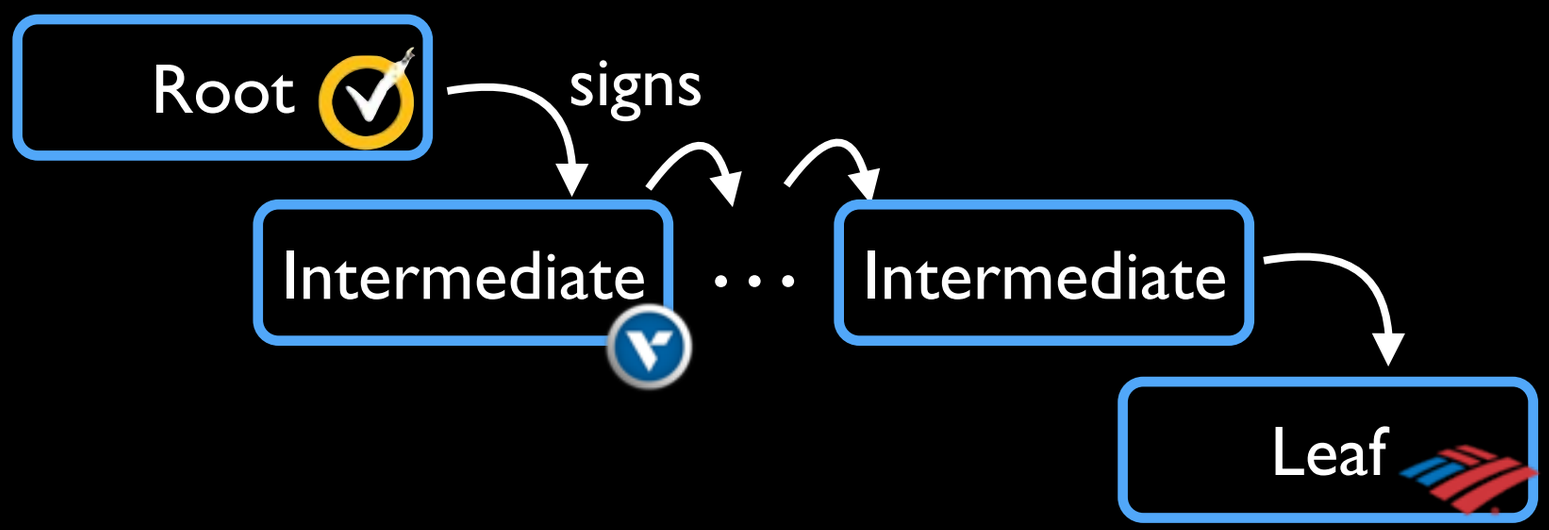
- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

Availability of revocation info

- Browsers *should* reject certs they cannot check
  - E.g., because the OCSP server is down

Chain lengths

- Browsers *should* reject a cert if *any* on the chain fail
  - Leaf, intermediate(s), root



Testing browser behavior

Revocation protocols

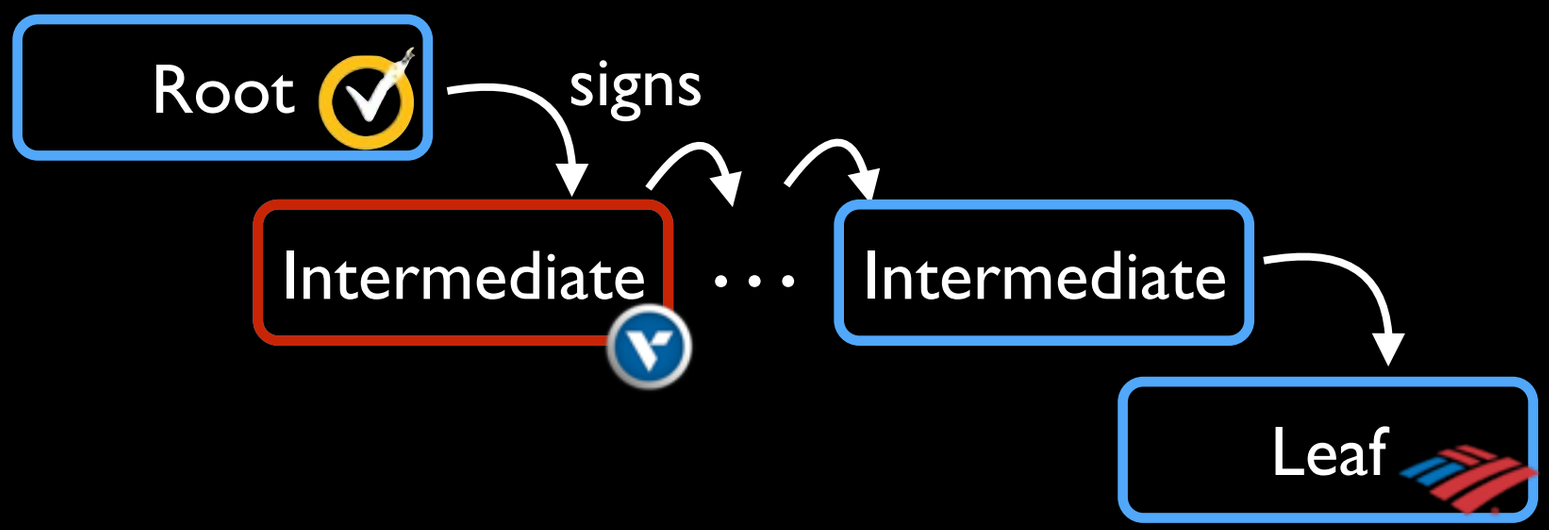
- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

Availability of revocation info

- Browsers *should* reject certs they cannot check
  - E.g., because the OCSP server is down

Chain lengths

- Browsers *should* reject a cert if *any* on the chain fail
  - Leaf, intermediate(s), root



Testing browser behavior

Revocation protocols

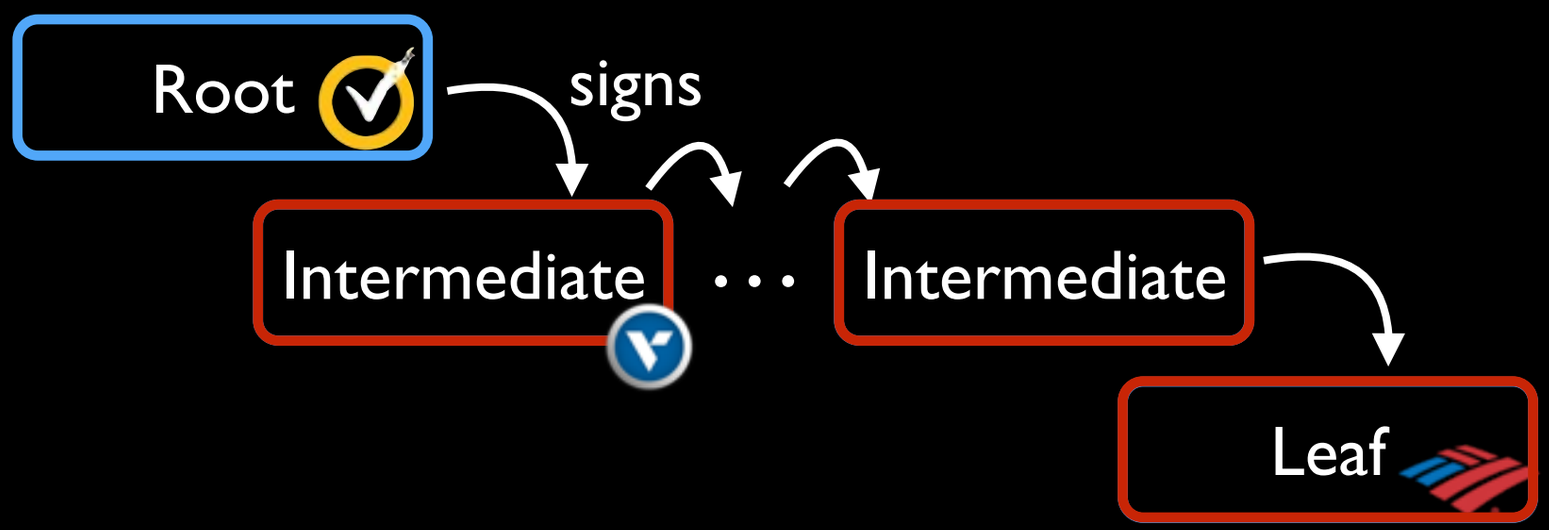
- Browsers *should* support all major protocols
  - CRLs, OCSP, OCSP stapling

Availability of revocation info

- Browsers *should* reject certs they cannot check
  - E.g., because the OCSP server is down

Chain lengths

- Browsers *should* reject a cert if *any* on the chain fail
  - Leaf, intermediate(s), root



Results across all browsers

		Desktop Browsers									Mobile Browsers			
		Chrome 42			Firefox	Opera		Safari	IE		iOS	Andr. 4.1–5.1		IE
		OS X	Win.	Linux	35–37	12.17	28.0	6–8	7–9	10–11	6–8	Stock	Chrome	8.0
	CRL													
Int. 1	Revoked	EV	✓	EV	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	EV	✓	–	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
Int. 2+	Revoked	EV	EV	EV	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Leaf	Revoked	EV	EV	EV	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	A	✗	✗	✗	✗
	OCSP													
Int. 1	Revoked	EV	EV	EV	EV	✗	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	L/W	✗	✓	✓	✗	✗	✗	✗
Int. 2+	Revoked	EV	EV	EV	EV	✗	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Leaf	Revoked	EV	EV	EV	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
	Unavailable	✗	✗	–	✗	✗	✗	✗	✗	A	✗	✗	✗	✗
	OCSP Stapling													
	Request OCSP Staple	✓	✓	✓	✓	✓	✓	✗	✓	✓	✗	I	I	✗
	Respect Revoked Staple	✗	✓	–	✓	✓	L/W	–	✓	✓	–	–	–	–

✔ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

A Pops up alert to user
L/W Passes on Linux/Win.

Results across all browsers

		Chrome 42 OS X	Chrome 42 Win.	Chrome 42 Linux
	CRL			
Int. 1	Revoked	EV	✓	EV
	Unavailable	EV	✓	–
Int. 2+	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
Leaf	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
	OCSP			
Int. 1	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
Int. 2+	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
Leaf	Revoked	EV	EV	EV
	Unavailable	✗	✗	–
	OCSP Stapling			
Request OCSP Staple		✓	✓	✓
Respect Revoked Staple		✗	✓	–

Chrome

Generally, only checks for EV certs
~3% of all certs

Allows if revocation info unavailable

Supports OCSP stapling

✔ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

A Pops up alert to user
L/W Passes on Linux/Win.

Results across all browsers

		Deskt Firefox 35–37
	CRL	
Int. 1	Revoked	✗
	Unavailable	✗
Int. 2+	Revoked	✗
	Unavailable	✗
Leaf	Revoked	✗
	Unavailable	✗
	OCSP	
Int. 1	Revoked	EV
	Unavailable	✗
Int. 2+	Revoked	EV
	Unavailable	✗
Leaf	Revoked	✓
	Unavailable	✗
	OCSP Stapling	
	Request OCSP Staple	✓
	Respect Revoked Staple	✓

Firefox

Never checks CRLs

Only checks intermediates for EV certs

Allows if revocation info unavailable

Supports OCSP stapling

✔ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

A Pops up alert to user
L/W Passes on Linux/Win.

Results across all browsers

		Safari 6–8
	CRL	
Int. 1	Revoked	✓
	Unavailable	✓
Int. 2+	Revoked	✓
	Unavailable	✗
Leaf	Revoked	✓
	Unavailable	✗
	OCSP	
Int. 1	Revoked	✓
	Unavailable	✗
Int. 2+	Revoked	✓
	Unavailable	✗
Leaf	Revoked	✓
	Unavailable	✗
	OCSP Stapling	
	Request OCSP Staple	✗
	Respect Revoked Staple	–

Safari

Checks CRLs and OCSP

Allows if revocation info unavailable
Except for first intermediate, for CRLs

Does *not* support OCSP stapling

✔ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

A Pops up alert to user
L/W Passes on Linux/Win.

Results across all browsers

		IE 7–9	IE 10–11
	CRL		
Int. 1	Revoked	✓	✓
	Unavailable	✓	✓
Int. 2+	Revoked	✓	✓
	Unavailable	✗	✗
Leaf	Revoked	✓	✓
	Unavailable	✗	A
	OCSP		
Int. 1	Revoked	✓	✓
	Unavailable	✓	✓
Int. 2+	Revoked	✓	✓
	Unavailable	✗	✗
Leaf	Revoked	✓	✓
	Unavailable	✗	A
	OCSP Stapling		
	Request OCSP Staple	✓	✓
	Respect Revoked Staple	✓	✓

Internet Explorer

Checks CRLs *and* OCSP

Often rejects if revocation info unavailable
Pops up alert for leaf in IE 10+

Supports OCSP stapling

✔ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

A Pops up alert to user
L/W Passes on Linux/Win.

Results across all browsers

		Mobile Browsers			
		iOS 6–8	Andr. 4.1–5.1 Stock	Chrome	IE 8.0
	CRL				
Int. 1	Revoked	✗	✗	✗	✗
	Unavailable	✗	✗	✗	✗
Int. 2+	Revoked	✗	✗	✗	✗
	Unavailable	✗	✗	✗	✗
Leaf	Revoked	✗	✗	✗	✗
	Unavailable	✗	✗	✗	✗
	OCSP				
Int. 1	Revoked	✗	✗	✗	✗
	Unavailable	✗	✗	✗	✗
Int. 2+	Revoked	✗	✗	✗	✗
	Unavailable	✗	✗	✗	✗
Leaf	Revoked	✗	✗	✗	✗
	Unavailable	✗	✗	✗	✗
	OCSP Stapling				
Request OCSP Staple		✗	I	I	✗
Respect Revoked Staple		–	–	–	–

Mobile Browsers

Uniformly *never* check

Android browsers request Staple
…and promptly ignore it

✔ Passes test
✗ Fails test

EV Passes for EV certs
I Ignores OCSP Staple

A Pops up alert to user
L/W Passes on Linux/Win.