CSE 127: Computer Security

Asymmetric Crypto, TLS, PKI and CT

Deian Stefan

Adopted slides from Kirill Levchenko and Dan Boneh

Asymmetric Cryptography

- Also called public key cryptography
- Two separate keys
  - Public key: known to everyone
  - Private key: used to decrypt and sign

Asymmetric Primitives

- Encryption and decryption
- Signing and verification
- Diffie Hellman key exchange

Asymmetric Keys

- Each user has a public and private key
- Keys related to each other in algorithm-dependent way
  - Need a key generation function
  - Keygen(r) = (pk, sk)
    - pk: public key
    - sk: secret key
    - r: random bits

Public-key encryption

- **Encryption:** (public key, plaintext) → ciphertext
  - $E_{pk}(m) = c$
- **Decryption:** (secret key, ciphertext) → plaintext
  - $D_{sk}(c) = m$

Encryption properties

- Encryption and decryption are inverse operations
  - $D_{sk}(E_{pk}(m)) = m$
- Secrecy: ciphertext reveals nothing about plaintext
  - Computationally hard to decrypt without secret key
- What's the point?
  - Anybody with your public key can send you a secret message!

Implementations

- ElGamal encryption (1985)
  - Based on Diffie-Helman key exchange (1976), itself invented by Diffie, Hellman, and Merkle
  - Computational basis: hardness of discrete logarithms
- RSA encryption (1978)
  - Invented by Rivest, Shamir, and Adleman
  - Computational basis: hardness of factoring

Digital signatures

- **Signing:** (secret key, message) → signature
  - $S_{sk}(m) = s$
- **Verification:** (public key, message, signature) → bool
  - $V_{pk}(m,s) = \text{true} \mid \text{false}$

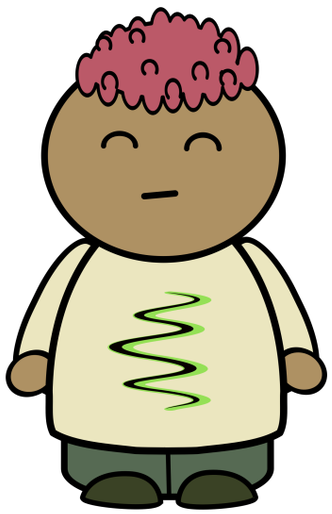
Signature properties

- Verification of signed message succeeds
  - $V_{pk}(m,S_{sk}(m))$ = true
- Unforgettability: can't compute signature for a message m without secret key sk
- What's the point?
  - Anybody with your public key can verify that you signed something!

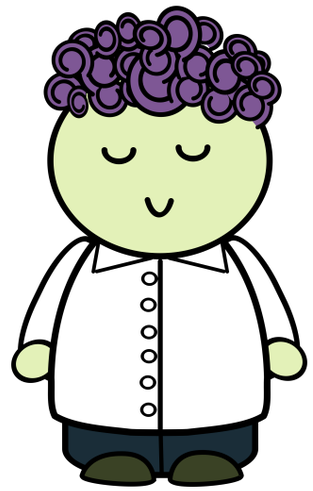
Implementations

- Digital Signature Algorithm (1991)
  - Closely related to ElGamal signature scheme (1984)
  - Computational basis: hardness of discrete logarithms
- RSA signatures
  - Invented by Rivest, Shamir, and Adleman
  - Computational basis: hardness of factoring

Encrypted and signed messaging



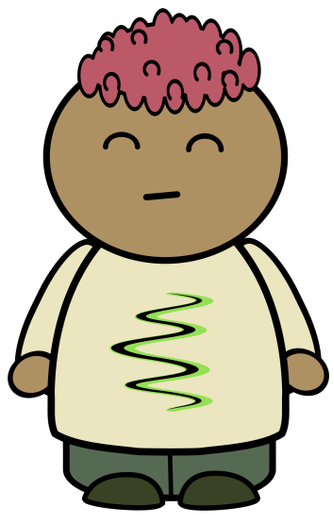
Alice



Bob

(this is a bad way to roll your own crypto)

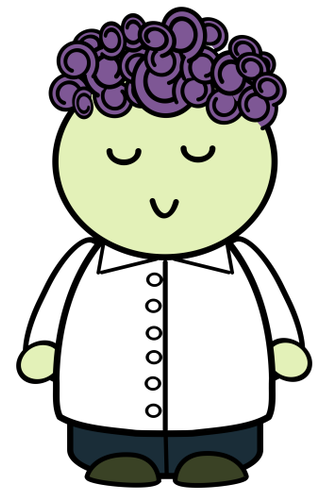
Encrypted and signed messaging



Alice

$(pk_{Alice\text{-}E}, sk_{Alice\text{-}E})$

$(pk_{Alice\text{-}S}, sk_{Alice\text{-}S})$



Bob

$(pk_{Bob\text{-}E}, sk_{Bob\text{-}E})$

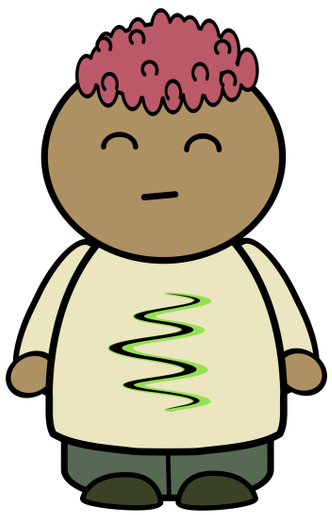
$(pk_{Bob\text{-}S}, sk_{Bob\text{-}S})$

(this is a bad way to roll your own crypto)

Encrypted and signed messaging

$E_{pkAlice\text{-}E}(m_1)=c_1$

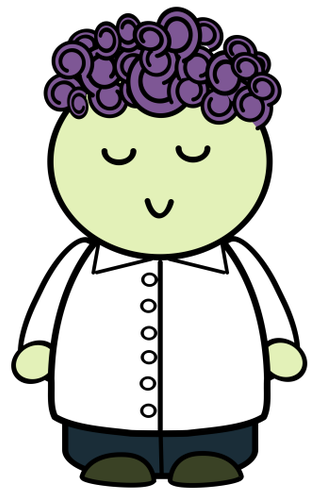
$(c_1, S_{skBob\text{-}S}(c_1))$



Alice

$(pk_{Alice\text{-}E}, sk_{Alice\text{-}E})$

$(pk_{Alice\text{-}S}, sk_{Alice\text{-}S})$



Bob

$(pk_{Bob\text{-}E}, sk_{Bob\text{-}E})$

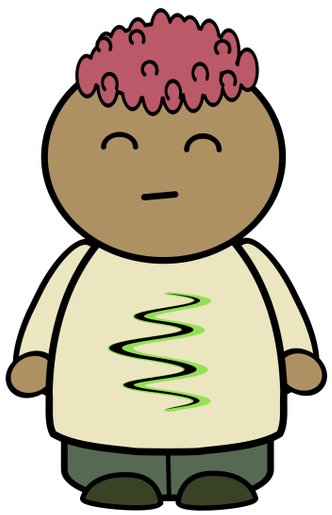
$(pk_{Bob\text{-}S}, sk_{Bob\text{-}S})$

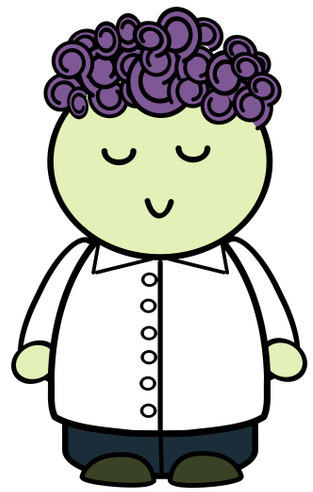
(this is a bad way to roll your own crypto)

Encrypted and signed messaging

$E_{pkAlice\text{-}E}(m_1)=c_1$

$(c_1, S_{skBob\text{-}S}(c_1))$





if $V_{pkBob\text{-}S}(c_1)$

$D_{skAlice\text{-}E}(c_1)$

Alice

$(pk_{Alice\text{-}E}, sk_{Alice\text{-}E})$

$(pk_{Alice\text{-}S}, sk_{Alice\text{-}S})$

Bob

$(pk_{Bob\text{-}E}, sk_{Bob\text{-}E})$

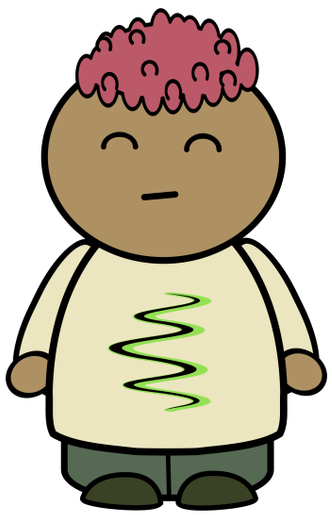
$(pk_{Bob\text{-}S}, sk_{Bob\text{-}S})$

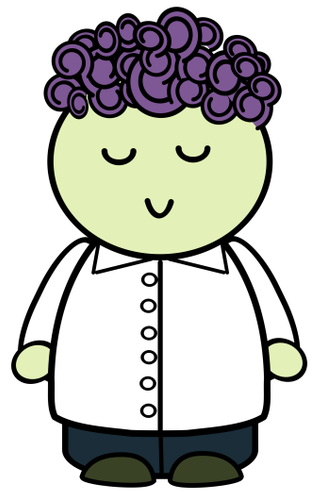
(this is a bad way to roll your own crypto)

Encrypted and signed messaging

$E_{pkAlice\text{-}E}(m_1)=c_1$

$(c_1, S_{skBob\text{-}S}(c_1))$





if $V_{pkBob\text{-}S}(c_1)$

$D_{skAlice\text{-}E}(c_1)$

Alice

$(pk_{Alice\text{-}E}, sk_{Alice\text{-}E})$

$(pk_{Alice\text{-}S}, sk_{Alice\text{-}S})$

Bob

$(pk_{Bob\text{-}E}, sk_{Bob\text{-}E})$

$(pk_{Bob\text{-}S}, sk_{Bob\text{-}S})$

(this is a bad way to roll your own crypto)

Practical Considerations

- Asymmetric cryptography operations are much more expensive than symmetric operations
  - Even implementations based on elliptic curves!
  - Don't want to encrypt/sign huge messages
- Moreover: asymmetric primitives operate on fixed-size messages

What do we do in practice?

- Usually combined with symmetric for performance
  - Use asymmetric to bootstrap ephemeral secret

Typical Encryption Usage

- Encryption:
  - Generate a ephemeral (one time) symmetric secret key
  - Encrypt message using ephemeral secret key
  - Encrypt ephemeral key using asymmetric encryption
- Decryption:
  - Decrypt ephemeral key, decrypt message

Typical Signature Usage

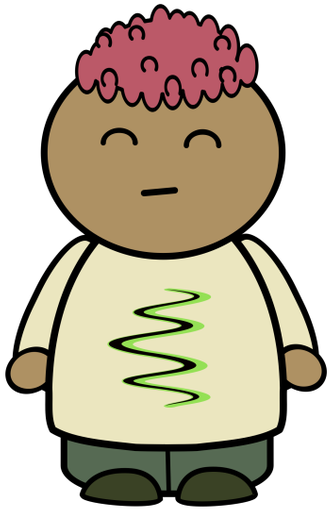
- Signing:
  - Compute cryptographic hash of message
  - Sign it using asymmetric signature scheme
- Verification:
  - Compute cryptographic hash of message
  - Verify it using asymmetric signature scheme

Asymmetric Primitives

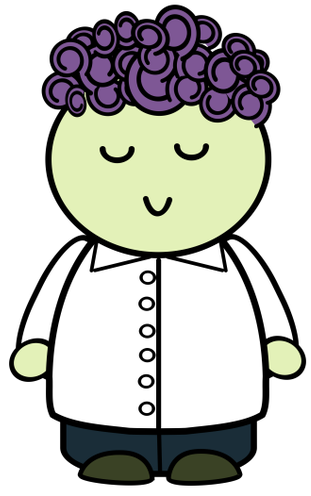
- Encryption and decryption
- Signing and verification
- Diffie Hellman key exchange

Diffie Hellman key exchange

- Establish a shared secret over public channel
  - Invented by Diffie, Hellman, and Merkle



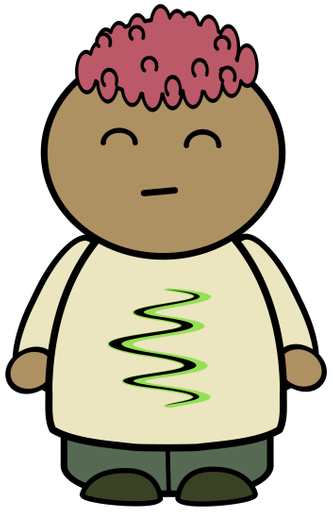
Alice



Bob

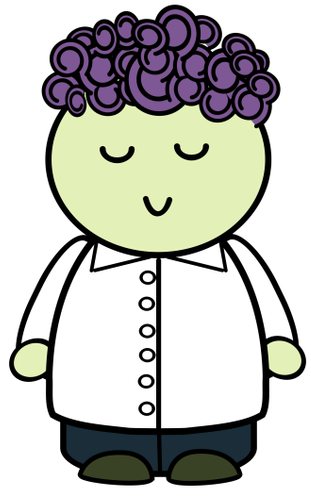
Diffie Hellman key exchange

- Establish a shared secret over public channel
  - Invented by Diffie, Hellman, and Merkle



Alice

$(pk_{Alice}, sk_{Alice}) \leftarrow DHKeygen()$

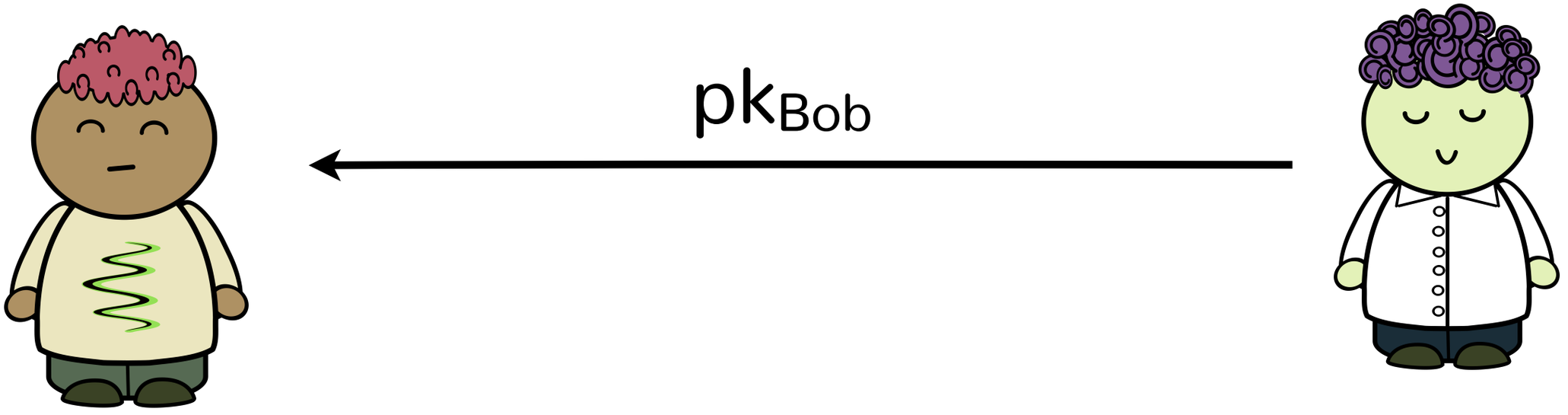


Bob

$(pk_{Bob}, sk_{Bob}) \leftarrow DHKeygen()$

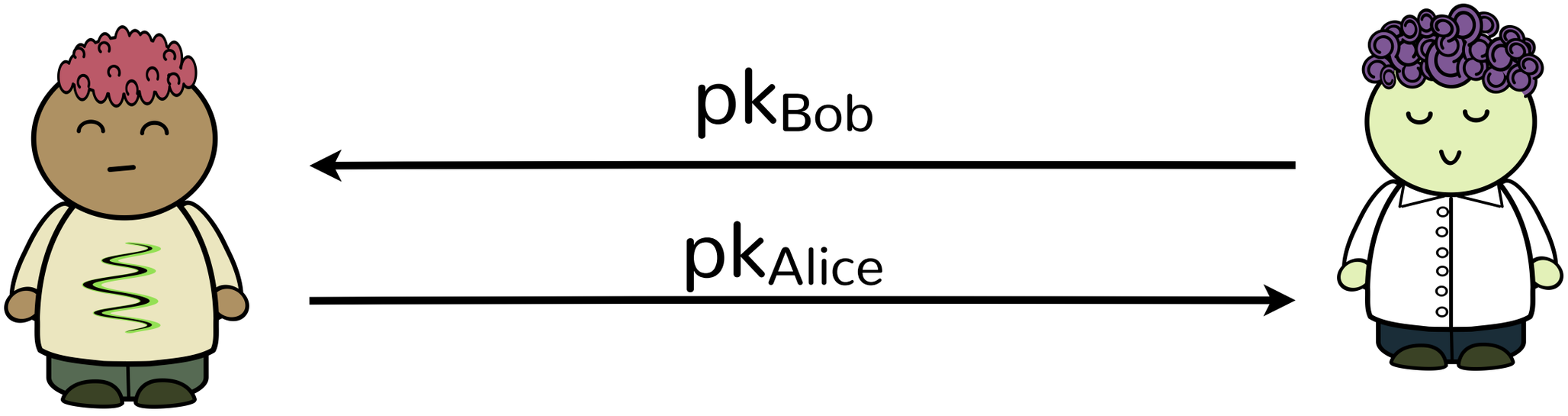
Diffie Hellman key exchange

- Establish a shared secret over public channel
  - Invented by Diffie, Hellman, and Merkle



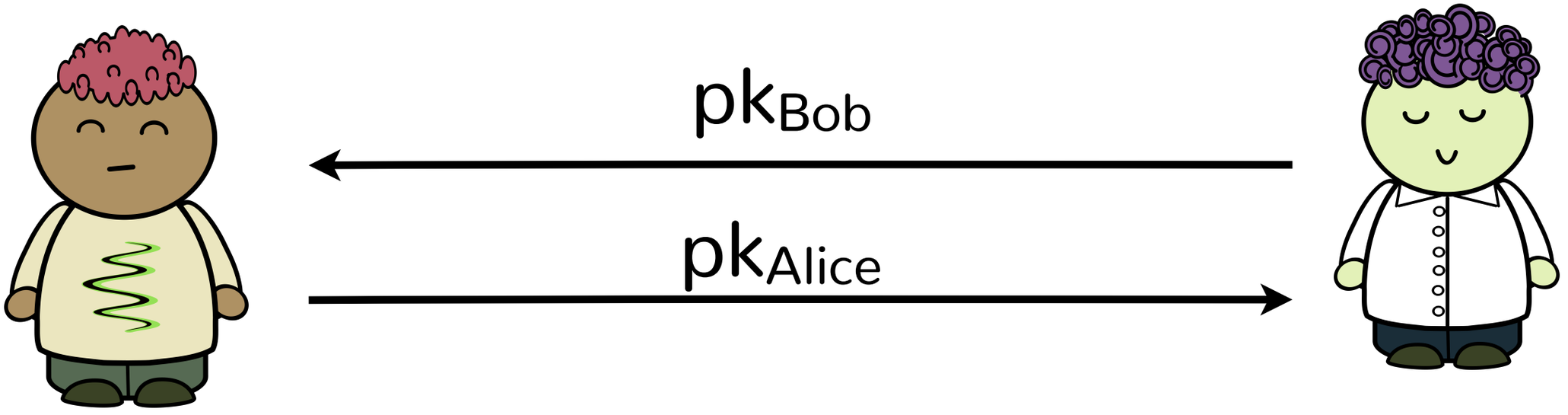
Diffie Hellman key exchange

- Establish a shared secret over public channel
  - Invented by Diffie, Hellman, and Merkle



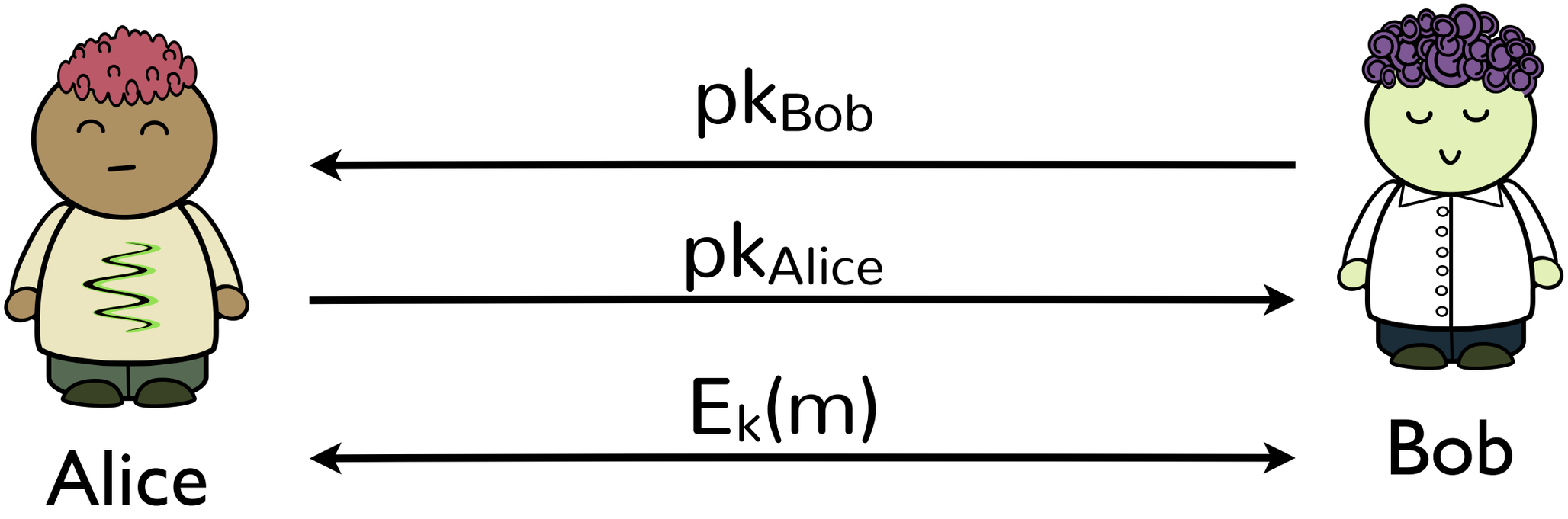
Diffie Hellman key exchange

- Establish a shared secret over public channel
  - Invented by Diffie, Hellman, and Merkle



Diffie Hellman key exchange

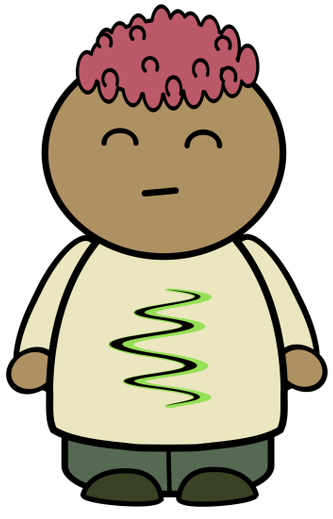
- Establish a shared secret over public channel
  - Invented by Diffie, Hellman, and Merkle



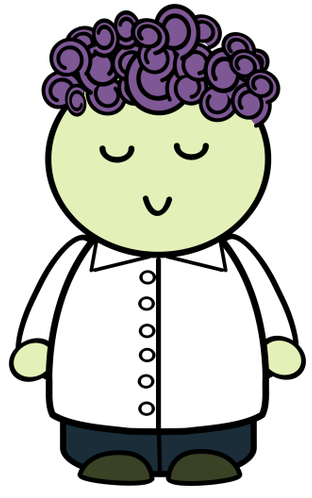
How do we get keys?

How do we get keys?

- Public keys are public: just ask for them!



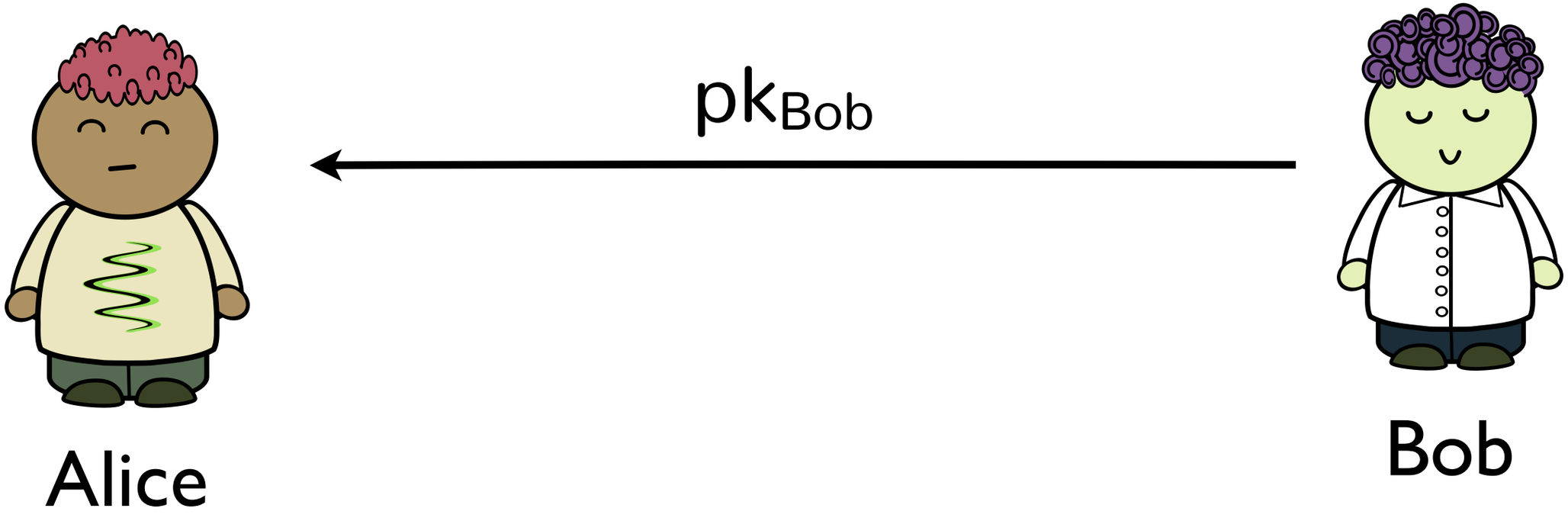
Alice



Bob

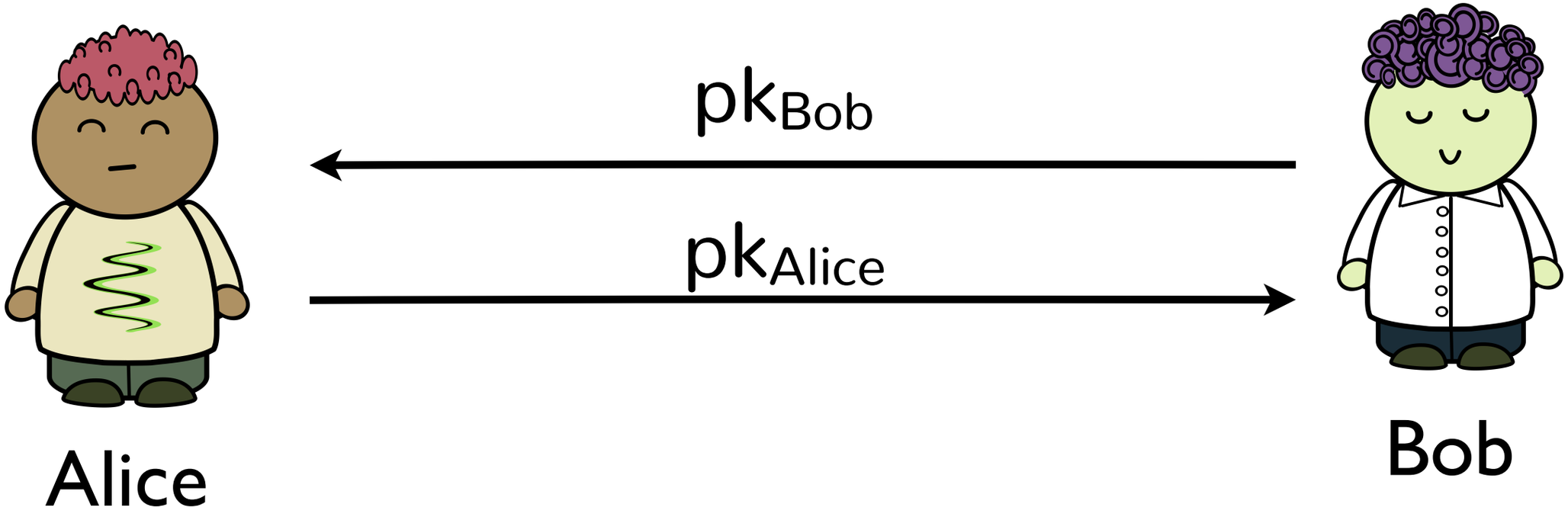
How do we get keys?

- Public keys are public: just ask for them!



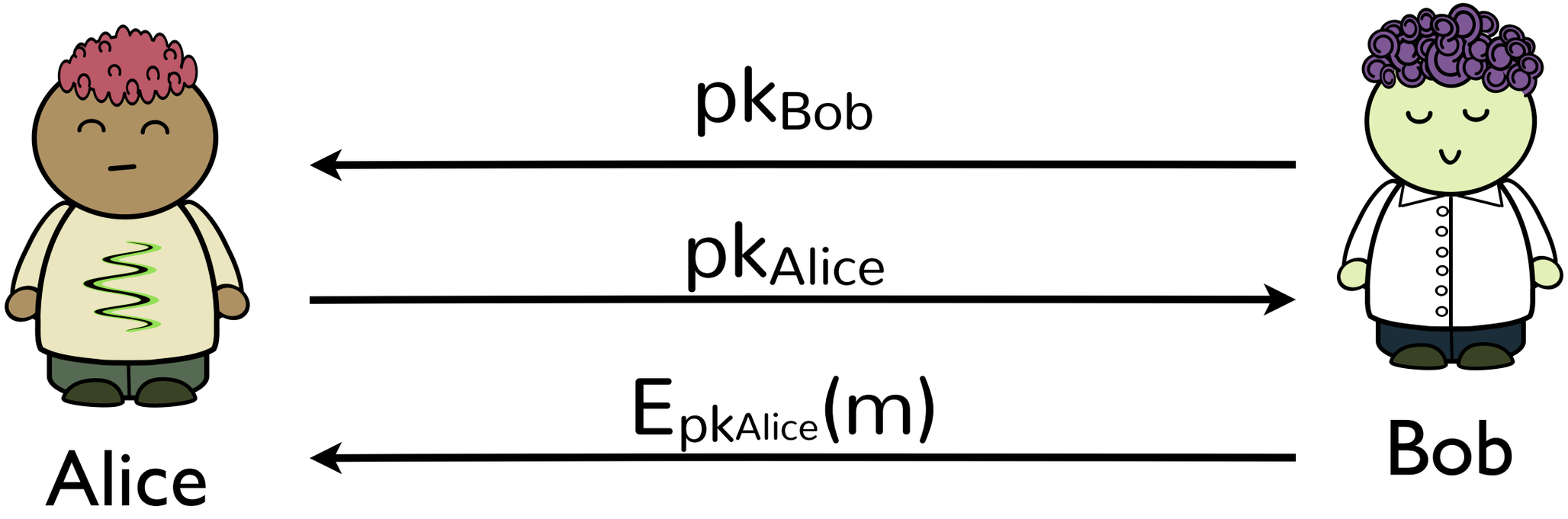
How do we get keys?

- Public keys are public: just ask for them!



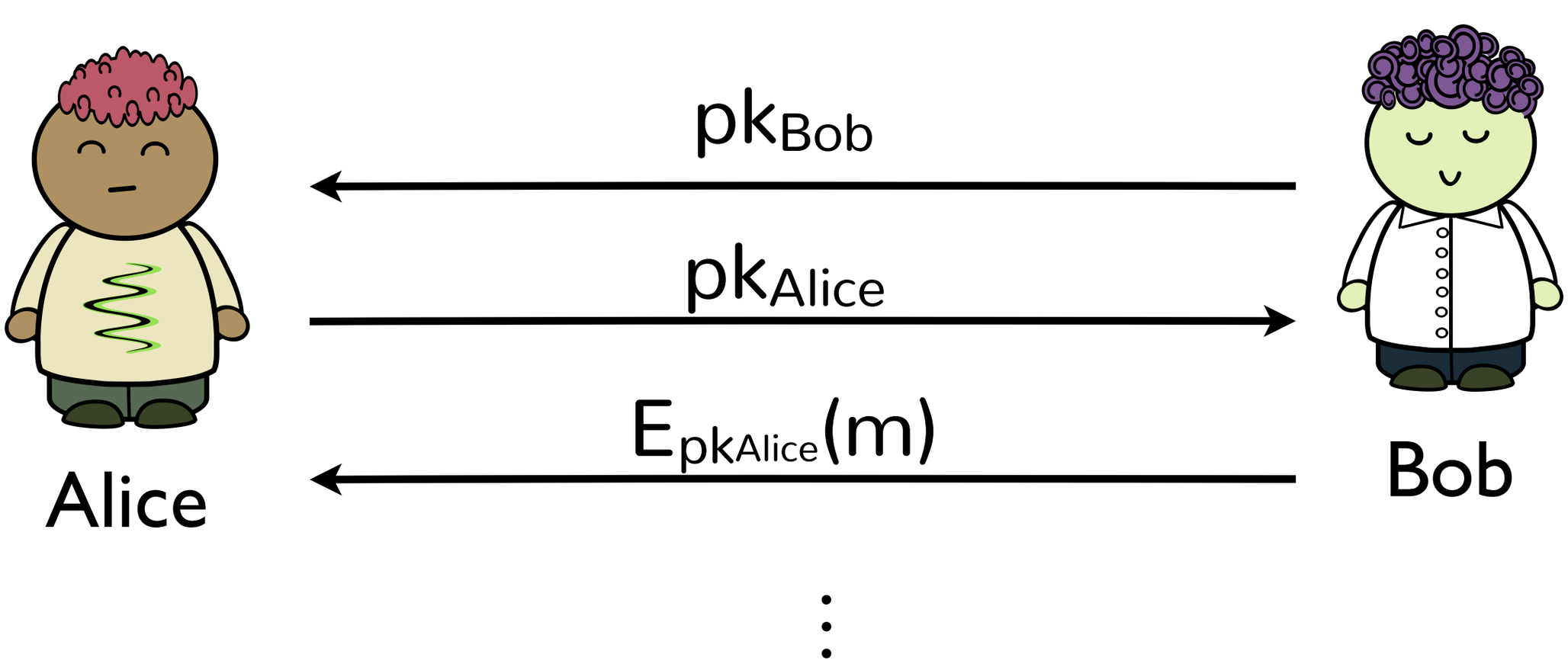
How do we get keys?

- Public keys are public: just ask for them!



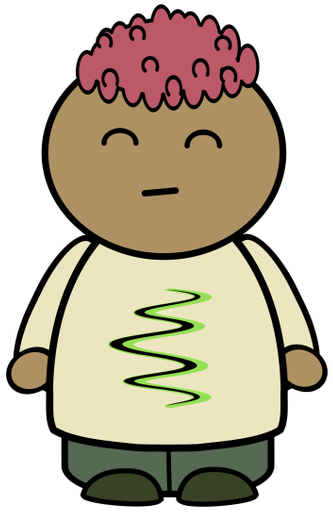
How do we get keys?

- Public keys are public: just ask for them!

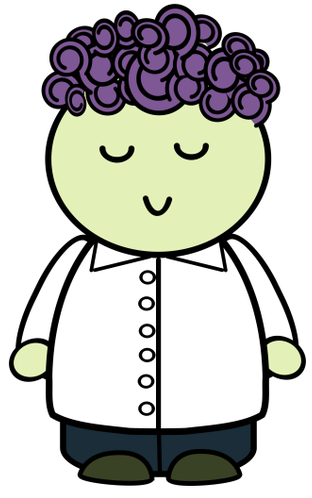


How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



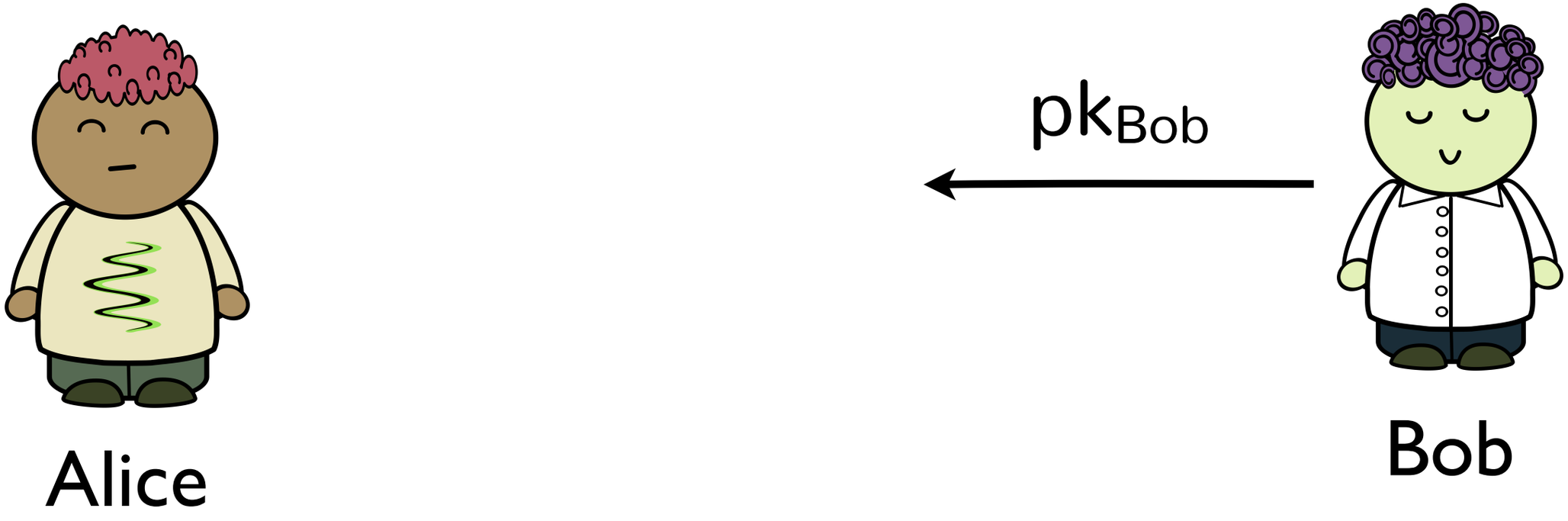
Alice



Bob

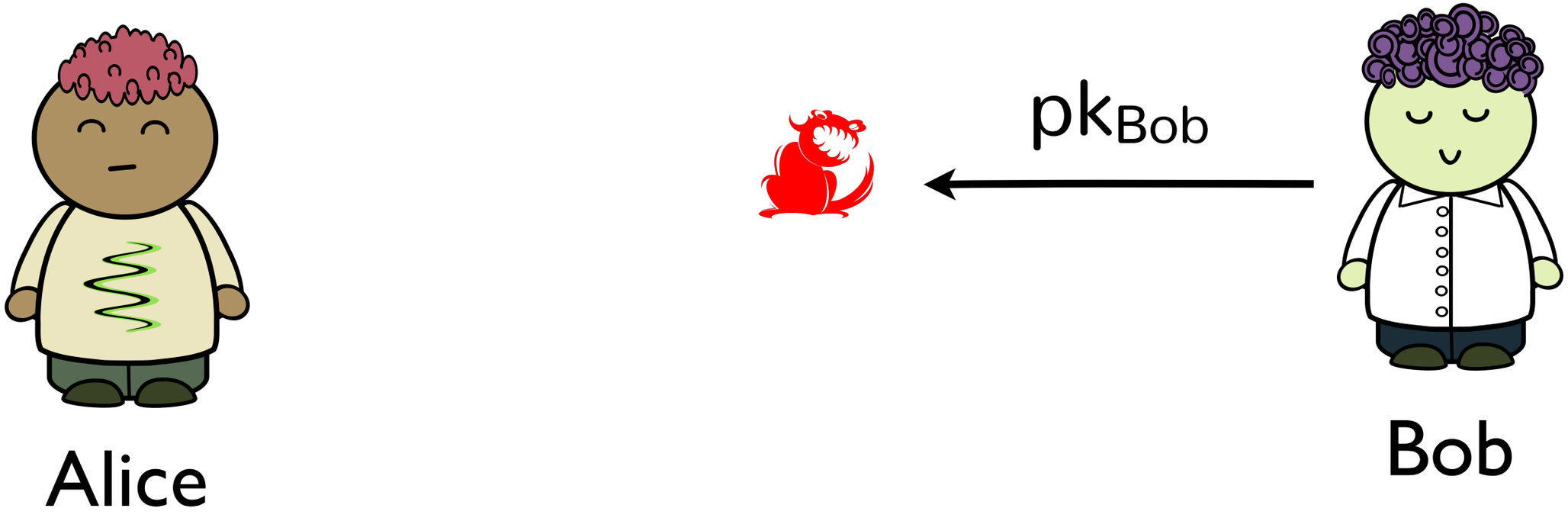
How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



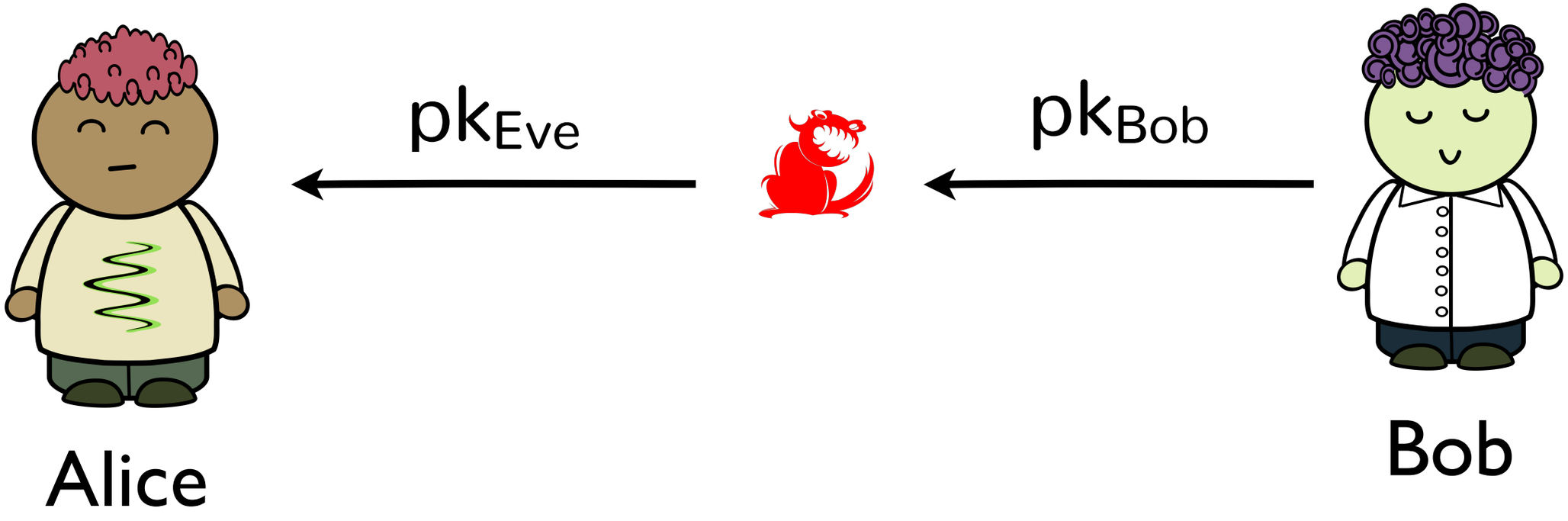
How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



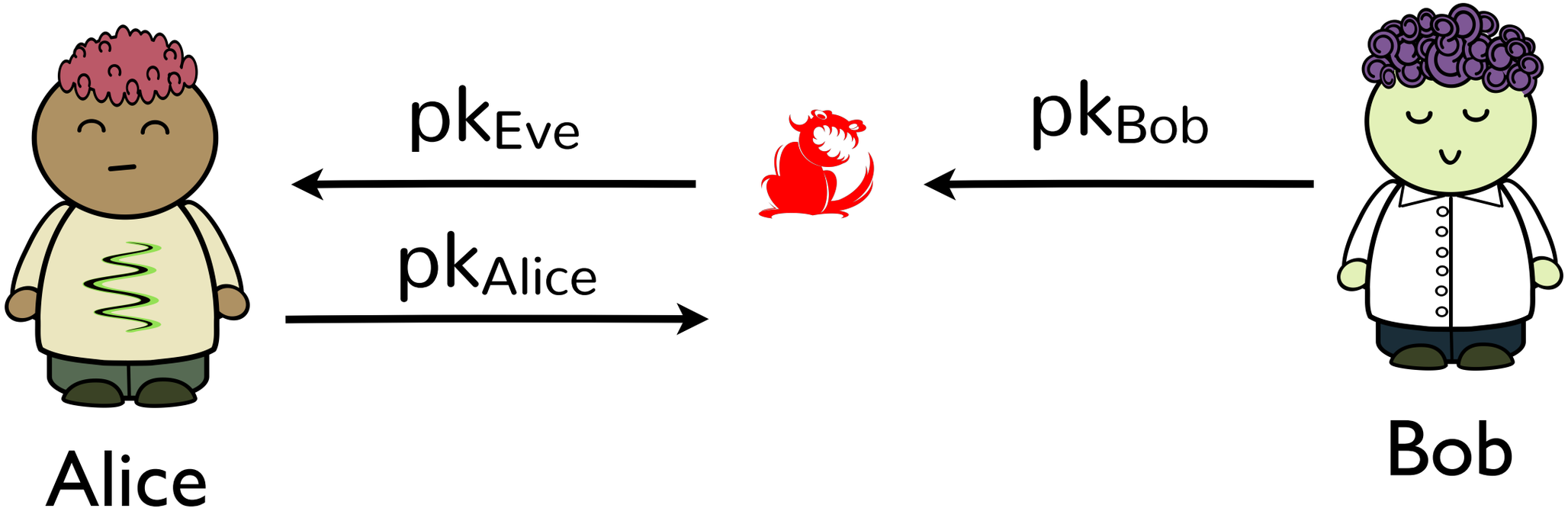
How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



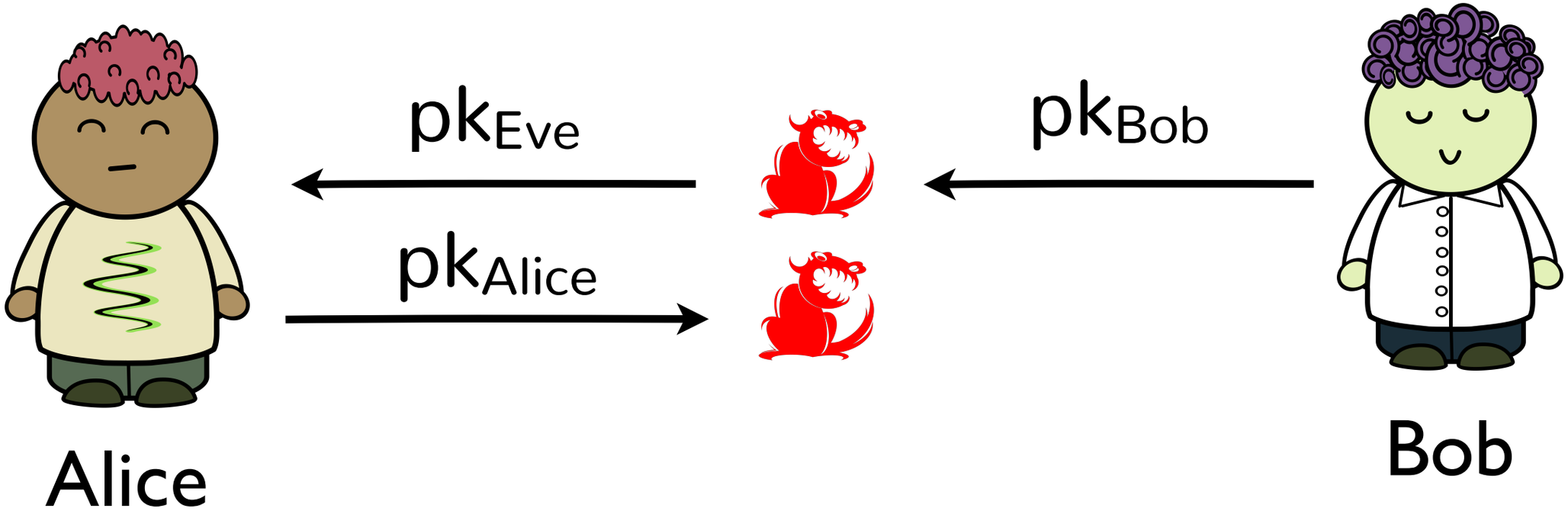
How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



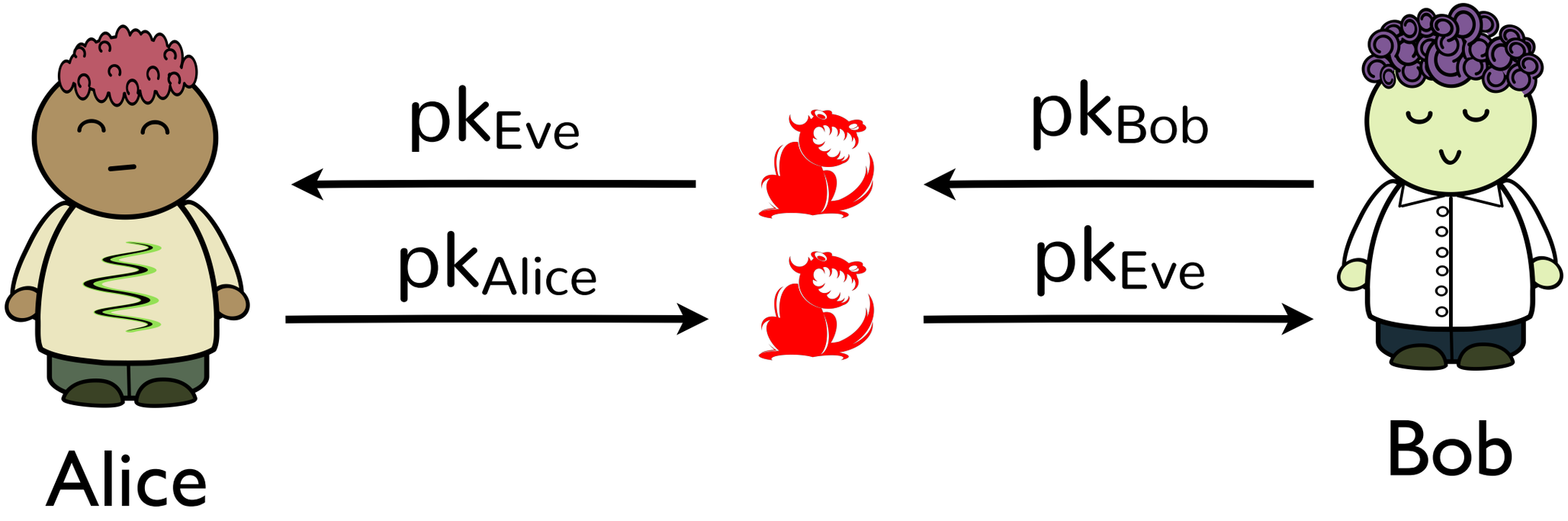
How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



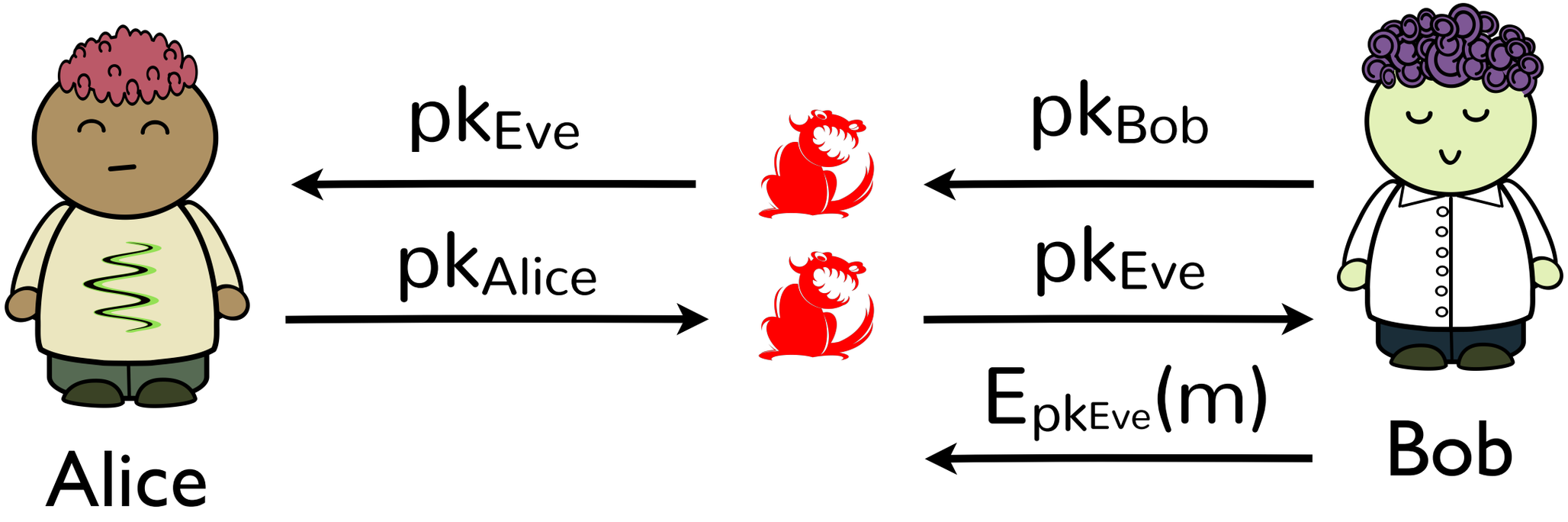
How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



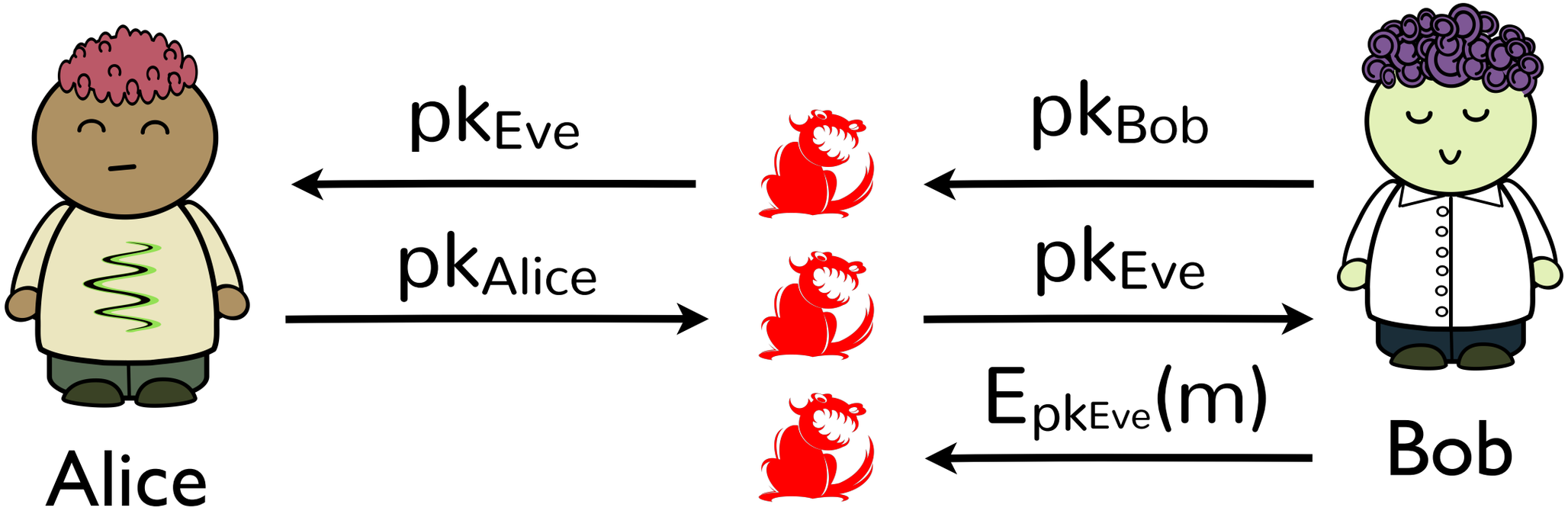
How do we get keys?

- Public keys are public: just ask for them!
  - ➤ No! Vulnerable to Man-in-the-Middle attacks!



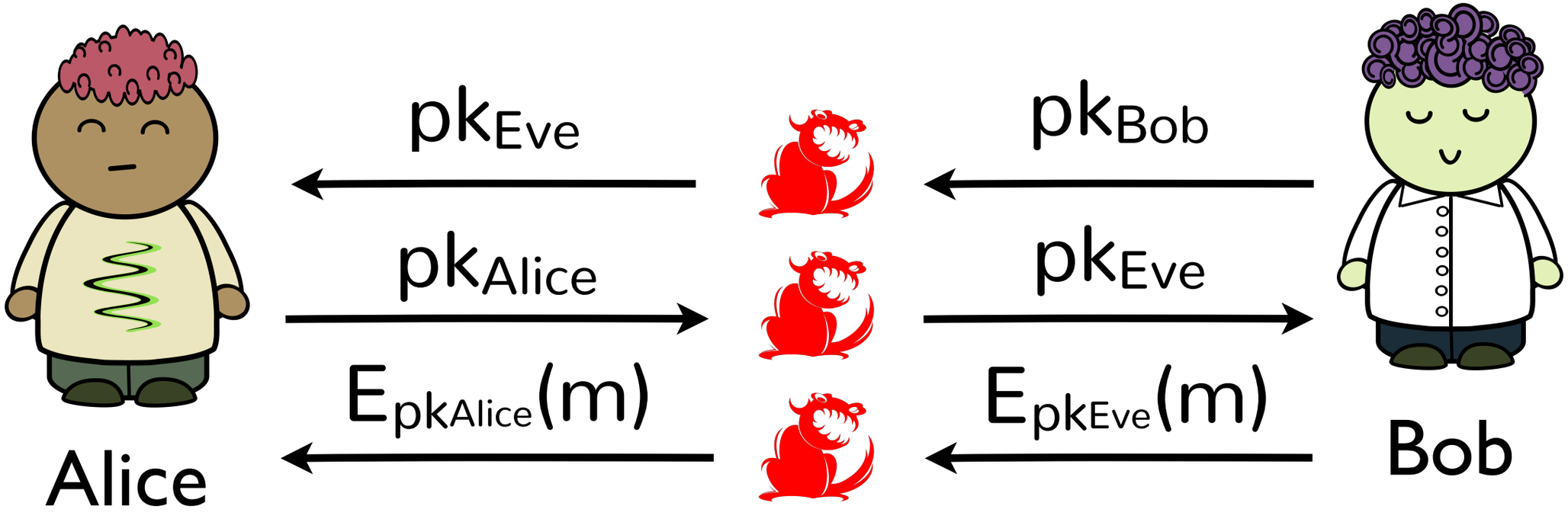
How do we get keys?

- Public keys are public: just ask for them!
  - ➤ No! Vulnerable to Man-in-the-Middle attacks!



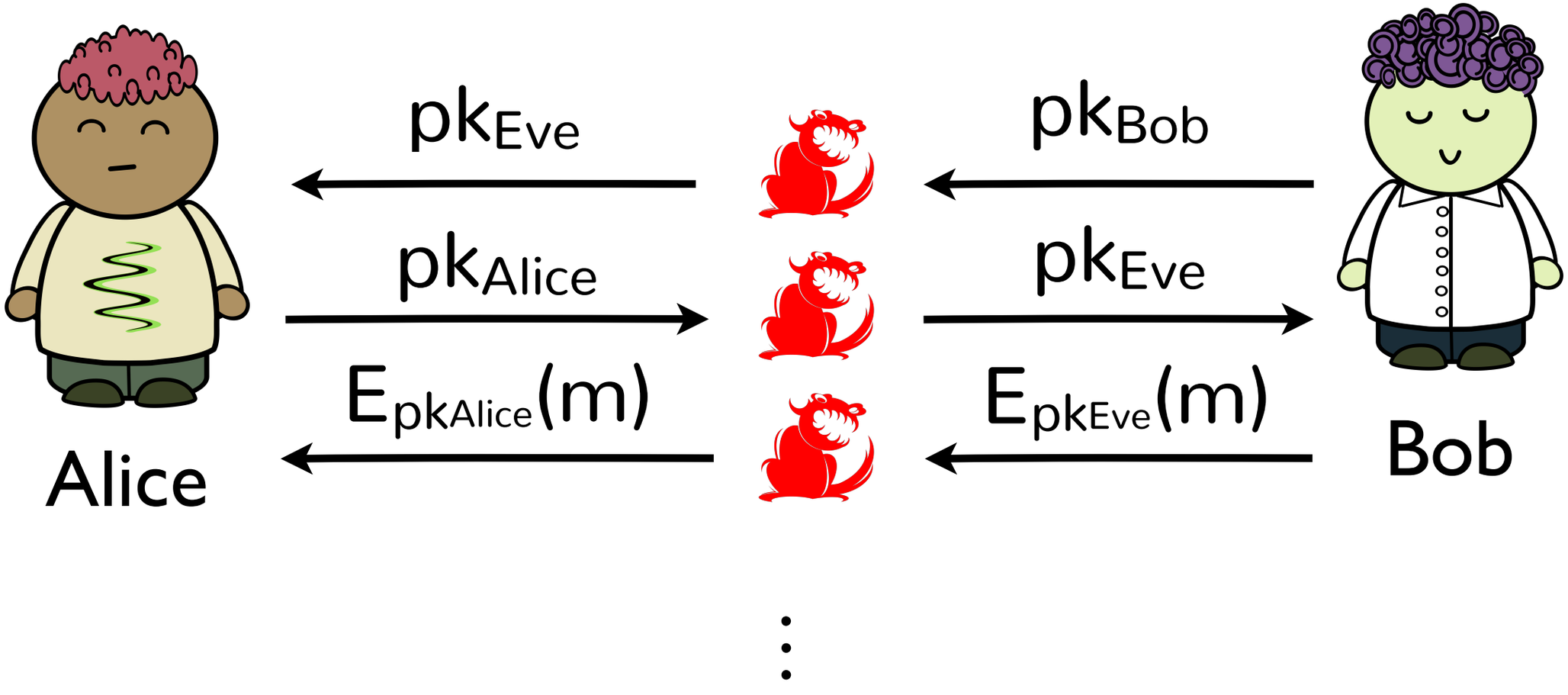
How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



How do we get keys?

- Public keys are public: just ask for them!
  - No! Vulnerable to Man-in-the-Middle attacks!



How do we get keys?

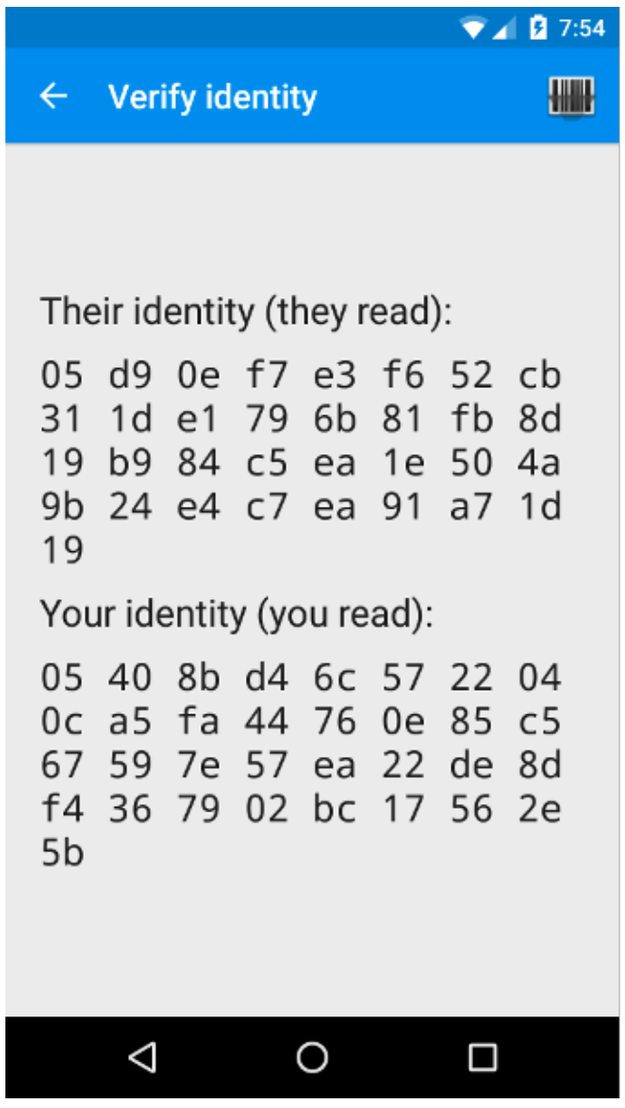
- Public directory contains everyone's public key
- To encrypt to a person, get their public key from directory
- No need for shared secrets!

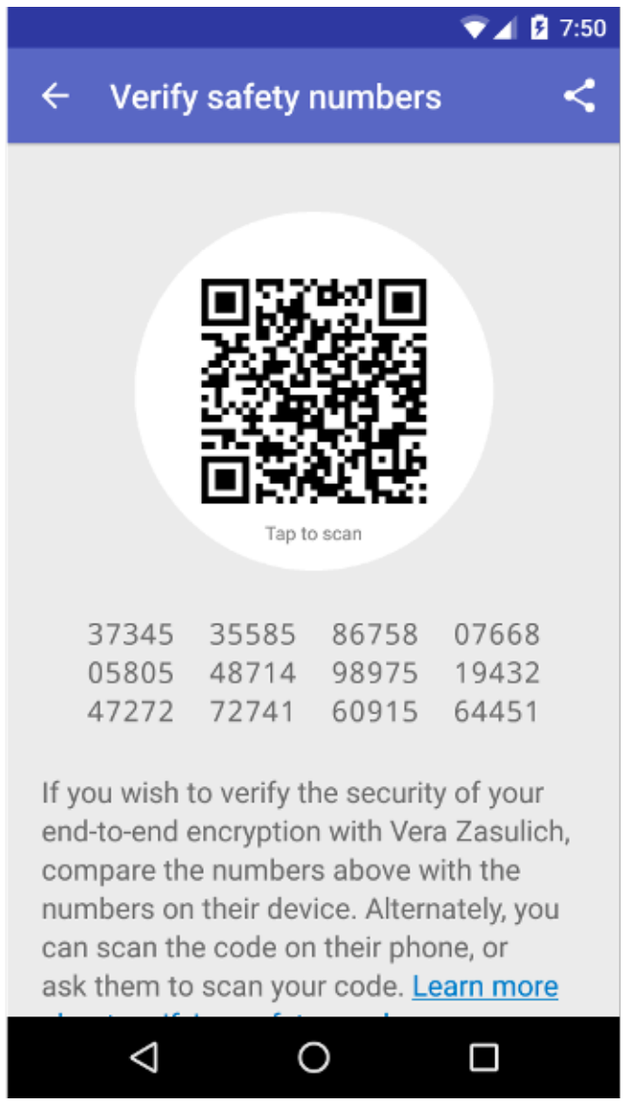
How do we know Alice’s key is actually Alice’s?

Key Verification

- Alice and Bob need a way to know that each has the real public key of the other
- **Ideal solution:** Alice and Bob meet in person and exchange public keys
- **Equivalent:** Alice and Bob meet in person and exchange public key fingerprints
  - Key fingerprint: cryptographic hash of public key
  - Key itself can be sent in the open

Where have you seen this?





https://signal.org/blog/safety-number-updates/

Key Verification

- **Problem with ideal:** Alice and Bob need to meet
  - Impractical to meet and verify key of everyone …
- **Practical solution:** Use a trusted intermediary
  - Alice and Bob have already exchanged keys w/ Claire
  - Claire sends signed message with Alice's key to Bob
  - Claire sends signed message with Bob's key to Alice
  - Alice and Bob trust Claire to send the public keys
  - Alice and Bob now have each other's public key

Key Verification Improved

- Claire creates a certificate:

  "I, Charlie, verified that Alice's key is … "

- Claire signs the message and gives it to Alice
  - Alice now has certificate attesting to her public key
- Alice sends Bob her public key and Claire's cert
- Bob verifies signature on certificate
- Bob trusts Claire, accepts public key from Alice

Who is Claire?

- PGP world: Claire is any other person you trust
- Keybase world: Claire is a set of
- SSL world: Claire is a Certificate Authority

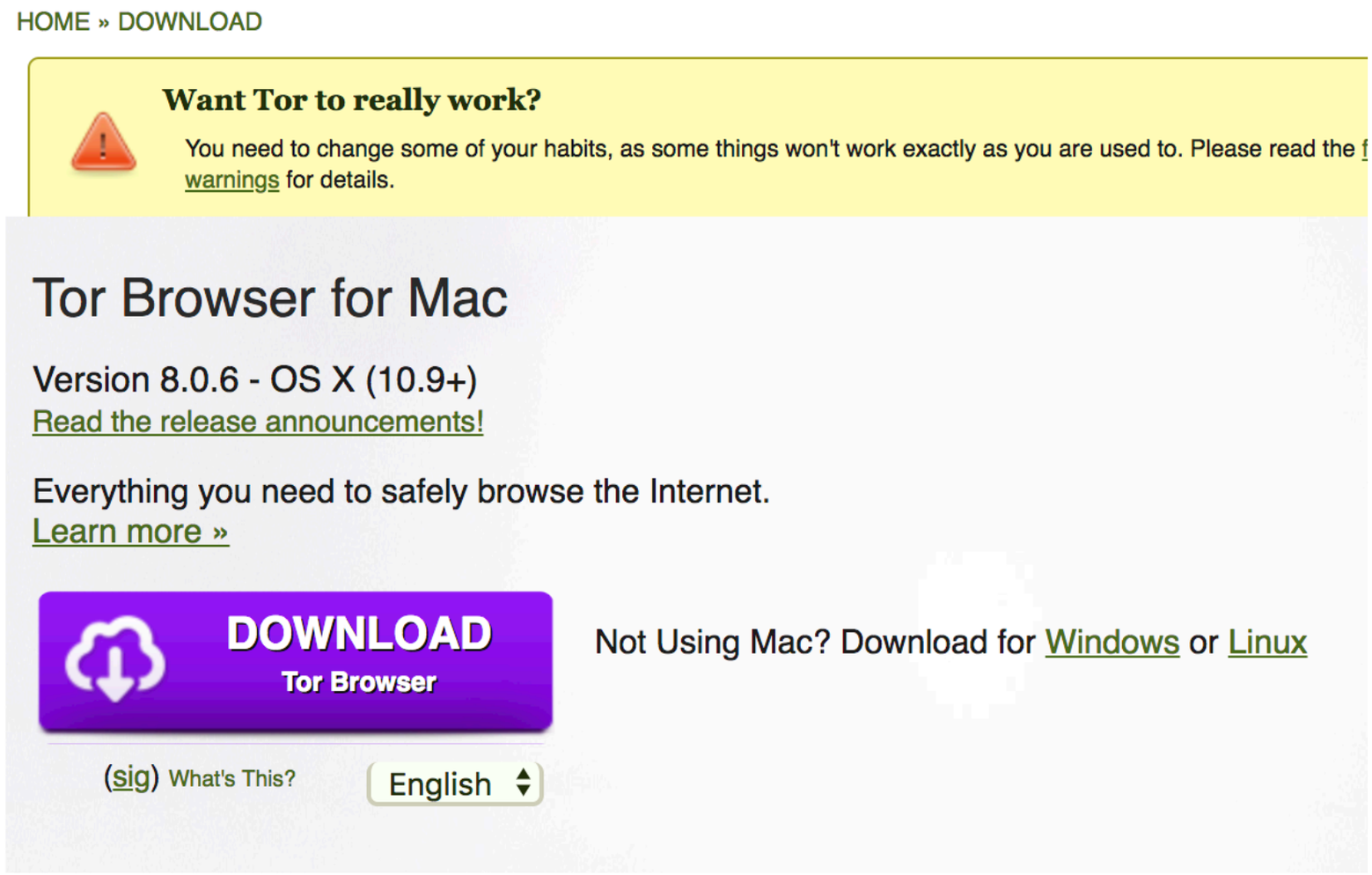
PGP (Web of Trust)

- PGP allows one user to attest to the accuracy of another user's public key — **key signing**
  - PGP does not use the term "certificate"
  - Public key has set of signatures (equiv. certificates)
- A user can indicate how much they trust another user's signature on a key

PGP (Web of Trust)

- Claire's signature of Bob's PGP key means Claire has verified that this is really Bob's key
  - Email address associated with key is Bob's address
  - Name associated with key is Bob
- Other people who trust Claire can use her signature on Bob's key to be sure it is Bob's key

Where is PGP used?



Where is PGP used?

HOME » DOWNLOAD

Want Tor to really work?

You need to change some of your habits, as some things won't work exactly as you are used to. Please read the f
warnings for details.

Tor Browser for Mac

Version 8.0.6 - OS X (10.9+)
Read the release announcements!

Everything you need to safely browse the Internet.
Learn more »

DOWNLOAD
Tor Browser

Not Using Mac? Download for Windows or Linux

(sig) What's This? English

```
-----BEGIN PGP SIGNATURE-----

iQIcBAABCgAGBQJcYsngAAoJEOt3RJHZ/wbir4kP/je6knpATcXLMrzcQD1kELx6
3fZTKDzW8KoXMRgu2XHngMuaRX8Vup/7xiTZ55URADadAh8yta5phvqt0OblfSRt
M92Yu12m0x5SAAzIbFCMnG0jvz7Lbt6LM2SuvL4mw0ifH4e3OshQqNNpz9e+C1UI
ShHNHlHo3N0A4ffYeNVSNjreLInjlJbsF5ZUFaznbFJlh5nvcDPSBCPt13M5bEwW
bjH/n+0tefVfhBKVjfRfLwEytjr7UbDO//8cz0q7du7wR4uB7iXgwNyEnh33kCaL
Ah45Qe3p9vY285XRTyYn3bjmXhg8p3DkLCzZ4l1TCYvZON0bJr84WxfIrVFJnbjJ
UtS+b0y8t24uhd9imrj6YdNbKrygQMSAT1LamUzwt03b2Hb6aTE1l5VK3k5DVzoj
4oyWmYcxX3qeQSfG8CdGW5iraK2bW0AkHTCdVWJuHByBzDTZ0774d06l4dy9zM+6
c014yhwmAyjQSFCZaui+1BrLSNWgnV4UNuoN9nI04NBVAin7/klBQiLJQ0Y/ZeKw
MPCJDj8jz64FtDg9IhMnimIHly7DtzItRbF2Mz0LCuU8ELc3ye+wJJO5QXGCS9Nx
dOEwIPSt8fnLh4KWABthceIXSk8OcnEnqMq6co7+x5hVRDbpnkkhv8zKXzmAKkKQ
AvpZ2rG1FA8GeUdFkres
=zXI0
-----END PGP SIGNATURE-----
```

Where is PGP used?

HOME » DOWNLOAD

Want Tor to really work?

You need to change some of your habits, as some things won't work exactly as you are used to. Please read the f
warnings for details.

Tor Browser for Mac

Version 8.0.6 - OS X (10.9+)
Read the release announcements!

Everything you need to safely browse the Internet.
Learn more »

DOWNLOAD
Tor Browser

Not Using Mac? Download for Windows or Linux

(sig)

```
-----BEGIN PGP SIGNATURE-----

iQIcBAABCgAGBQJcYsngAAoJEOt3RJHZ/wbir4kP/je6knpATcXLMrzcQD1kELx6
3fZTKDzW8KoXMRgu2XHngMuaRX8Vup/7xiTZ55URADadAh8yta5phvqt0OblfSRt
M92Yu12m0x5SAAzIbFCMnG0jvz7Lbt6LM2SuvL4mw0ifH4e3OshQqNNpz9e+C1UI
ShHNHlHo3N0A4ffYeNVSNjreLInjlJbsF5ZUFaznbFJlh5nvcDPSBCPt13M5bEwW
bjH/n+0tefVfhBKVjfRfLwEytjr7UbDO//8cz0q7du7wR4uB7iXgwNyEnh33kCaL
Ah45Qe3p9vY285XRTyYn3bjmXhg8p3DkLCzZ4l1TCYvZON0bJr84WxfIrVFJnbjJ
UtS+b0y8t24uhd9imrj6YdNbKrygQMSAT1LamUzwt03b2Hb6aTE1l5VK3k5DVzoj
4oyWmYcxX3qeQSfG8CdGW5iraK2bW0AkHTCdVWJuHByBzDTZ0774d06l4dy9zM+6
c014yhwmAyjQSFCZaui+1BrLSNWgnV4UNuoN9nI04NBVAin7/klBQiLJQ0Y/ZeKw
MPCJDj8jz64FtDg9IhMnimIHly7DtzItRbF2Mz0LCuU8ELc3ye+wJJO5QXGCS9Nx
dOEwIPSt8fnLh4KWABthceIXSk8OcnEnqMq6co7+x5hVRDbpnkkhv8zKXzmAKkKQ
AvpZ2rG1FA8GeUdFkres
=zXI0
-----END PGP SIGNATURE-----
```

Where is PGP used?

Where is PGP used?

pacman/Package signing

< Pacman

To determine if packages are authentic, *pacman* uses **GnuPG keys** in a **web of trust** model. The current Master Signing Keys are found **here**. At least three of these Master Signing Keys are used to sign each of the Developer's and Trusted User's own keys which then in turn are used to sign their packages. The user also has a unique PGP key which is generated when you set up *pacman-key*. So the web of trust links the user's key to the Master Keys.

Examples of webs of trust:

- **Custom packages**: You made the package yourself and signed it with your own key.
- **Unofficial packages**: A developer made the package and signed it. You used your key to sign that developer's key.
- **Official packages**: A developer made the package and signed it. The developer's key was signed by the Arch Linux master keys. You used your key to sign the master keys, and you trust them to vouch for developers.

Related articles

GnuPG

DeveloperWiki:Package signing

Where is PGP used?

archlinux™ Home Packages Forums **Wiki** Bugs Security AUR Download

Create account Log in

Main page
Table of contents
Getting involved
Wiki news
Random page

Interaction
Help
Contrib
Recent
New pa
Statisti
Reques

Tools
What li
Relatec
Specia
Printab

Page Discussion Read View source View history Search ArchWiki

pacman/Package signing

< Pacman

KEYRING

About Debian Getting Debian Support Developers' Corner

debian / keyring.debian.org

Debian Public Key Server

This public key server provides simple HKP *lookup* and *add* requests for Debian developer and maintainer public keys.

The server may be accessed with gpg by using the `--keyserver` option in combination with either of the `--recv-keys` or `--send-keys` actions.

Please note that this server is meant only for basic key retreive/update operation, and does not implement search functionality. To search for a specific D
Developer, use the Developer LDAP Search interface.

Only keys in the Debian keyrings (ie those for DDs and DMs) will be returned by this server and only pre-existing keys will be updated, although a copy
updates will be forwarded to the keyserver network.

You can use the keyring server for the following purposes:

Fetch a key

Once you know the key's ID, just ask the server for it:

Where is PGP used?

archlinux™

Main page
Table of contents
Getting involved
Wiki news
Random page

Interaction

Help
Contrib
Recent
New pa
Statisti
Reques

Tools

What li
Related
Specia
Printab

Page | Discussion

pacman/Package signing

< Pacman

KEYRING

About Debian | Getting Debian | Support | Developers' C

debian / keyring.debian.org

Debian Public Key Server

This public key server provides simple HKP *lookup* and *ad*

The server may be accessed with gpg by using the `--keyse`

Please note that this server is meant only for basic key retr
Developer, use the Developer LDAP Search interface.

Only keys in the Debian keyrings (ie those for DDs and DM
updates will be forwarded to the keyserver network.

You can use the keyring server for the following purposes:

Fetch a key

Once you know the key's ID, just ask the server for it:

7.4 Git Tools - Signing Your Work

Signing Your Work

Git is cryptographically secure, but it's not foolproof. If you're taking work from others on th
want to verify that commits are actually from a trusted source, Git has a few ways to sign a
using GPG.

GPG Introduction

First of all, if you want to sign anything you need to get GPG configured and your persona

```
$ gpg --list-keys
/Users/schacon/.gnupg/pubring.gpg
---------------------------------
pub   2048R/0A46826A 2014-06-04
uid                  Scott Chacon (Git signing key) <schacon@gmail.com>
sub   2048R/874529A9 2014-06-04
```

If you don't have a key installed, you can generate one with `gpg --gen-key`.

```
$ gpg --gen-key
```

Once you have a private key to sign with, you can configure Git to use it for signing things
`user.signingkey` config setting.

```
$ git config --global user.signingkey 0A46826A
```

Now Git will use your key by default to sign tags and commits if you want.

Signing Tags

If you have a GPG private key setup, you can now use it to sign new tags. All you have to

Where is PGP used?

Not really for email....

Why Johnny Still Can't Encrypt:
Evaluating the Usability of Email Encryption Software

Steve Sheng
Engineering and Public Policy
Carnegie Mellon University
shengx@cmu.edu

Levi Broderick
Electrical and Computer Engineering
Carnegie Mellon University
lpb@ece.cmu.edu

Colleen Alison Koranda
HCI Institute
Carnegie Mellon University
ckoranda@andrew.cmu.edu

Jeremy J. Hyland
Heinz School of Public Policy and Management
Carnegie Mellon University
jhyland@andrew.cmu.edu

Who is Claire?

- PGP world: Claire is any other person you trust
- Keybase world: Claire is a set of
- SSL world: Claire is a Certificate Authority

Keybase world

- Challenge: verifying Alice's key is actually Alice's key on public directory
- Solution: Associate key with social network identifies
  - Sign messages on GitHub, Twitter, Facebook, etc.
- Attacker model
  - Attacher must compromise N of my account

Keybase world

y is actually Alice's

social network

- Sign messages on GitHub, Twitter, Facebook, etc.

- Attacker model
  - Attacher must compromise N of my account

Deian Stefan
@deiandelmars

Verifying myself:
PItQadTXbIpi6Vg
K / keybase.io/d

Translate Tweet

2:41 PM - 2 Apr 2014

https://gist.github.com/deian/9906028

I hereby claim:

- I am deian on github.
- I am deian (https://keybase.io/deian) on keybase.
- I have a public key whose fingerprint is A3CA DAA1 144E 5CDE B67F 37B9 5ED1 79BB 628C 02E2

To claim this, I am signing this object:


```
{
    "body": {
        "key": {
            "fingerprint": "a3cadaa1144e5cdeb67f37b95ed179bb628c02e2",
            "host": "keybase.io",
            "key_id": "5ED179BB628C02E2",
            "uid": "56845ebd23c5b6f3004a3da75554ff00",
            "username": "deian"
        },
        "service": {
            "name": "github",
            "username": "deian"
        },
        "type": "web_service_binding",
        "version": 1
    },
    "ctime": 1396315705,
    "expire_in": 157680000,
    "prev": "70543d8dd91a406196f3addecd945da82b05094c5a38b899b0664aacd6bddb5b",
    "seqno": 2,
    "tag": "signature"
}
```


with the PGP key whose fingerprint is A3CA DAA1 144E 5CDE B67F 37B9 5ED1 79BB 628C 02E2 (captured `body.key.fingerprint`), yielding the PGP signature:


```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v2.0.22 (GNU/Linux)

owGbwMvMwMQYd7Fyd1IP0yPG0weeJDEEW4lZVisl5adUKllVK2Wngqm0zLz01KKC
osy8EiUrpUTj5MSUxERDQxOTVNPklNQkM/M0Y/MkS9PUFENzy6QkMyOLZAOjVCMl
HaWM/GKQDqAxSYnFqXqZ+UAxICc+MwUoaurqAlTv5ARU72xg5ApSXwqRMLMwMU1N
SjEyTjZNMkszNjAwSTROSTQ3NTU1SUszMAApLE4tykvMTQWqTknNTMxTqtVRAgqV
```


- Sign m … tc.
- Attacker …
- Attache …

Deian Stefan
@deiandelmars

Verifying myself:
PItQadTXbIpi6Vg
K / keybase.io/d

Translate Tweet

2:41 PM - 2 Ap

https://gist.github.com/deian/9906028

I hereby claim:

- I am deian on github.
- I am deian (https://keybase.io/deian) on keybase.
- I have a public key whose fingerprint is A3CA DAA1 144E 5CDE B67F 37B9 5ED1 79BB 628C 02E2

To claim this, I am signing this object:

```
{
    "body": {
        "key": {
            "fingerprint": "a3cadaa1144e5cdeb67f37b95ed179bb628c02e2",
            "host": "keybase.io",
            "key_id": "5ED179BB628C02E2",
            "uid": "56845ebd23c5b6f3004a3da75554ff00",
            "username": "deian"
        },
        "service": {
```

ce's

- A

Hacker News new | past | comments | ask | show | jobs | submit

user: deian
created: October 13, 2014
karma: 22
about: Ass. Prof at UCSD (Fall 2016). Chief Scientist at GitStar.

If you want to hack on compilers, run-time systems, or securing Node.js contact me (see how: http://deian.org).

[my public key: https://keybase.io/deian; my proof: https://keybase.io/deian/sigs/qJGyCaj7fvHQK977LrsMDAdUHgt_G1hWRBenW0sbK6E]

submissions
comments
favorites

```
Version: GnuPG v2.0.22 (GNU/Linux)

owGbwMvMwMQYd7Fyd1IP0yPG0weeJDEEW4lZVisl5adUKllVK2Wngqm0zLz01KKC
osy8EiUrpUTj5MSUxERDQxOTVNPklNQkM/M0Y/MkS9PUFENzy6QkMyOLZAOjVCMl
HaWM/GKQDqAxSYnFqXqZ+UAxICc+MwUoaurqAlTv5ARU72xg5ApSXwqRMLMwMU1N
SjEyTjZNMkszNjAwSTROSTQ3NTU1SUszMAApLE4tykvMTQWqTknNTMxTqtVRAgqV
```

Who is Claire?

- PGP world: Claire is any other person you trust
- Keybase world: Claire is a set of
- SSL world: Claire is a Certificate Authority

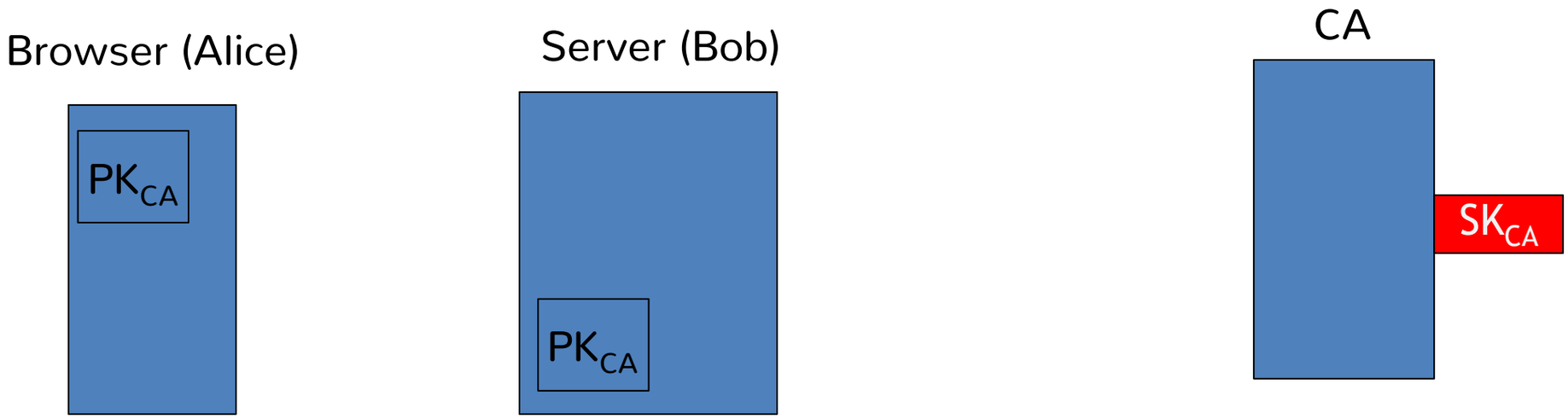
Announcements

- Last PA due Tue/Sat
- Final: March 20th
- Review: Monday 4PM in 1202
- Today: Rehash certificates
  - Then: A: General network security
    B: DNS security
    C: Review
  - Then: sphiel about ethics etc.

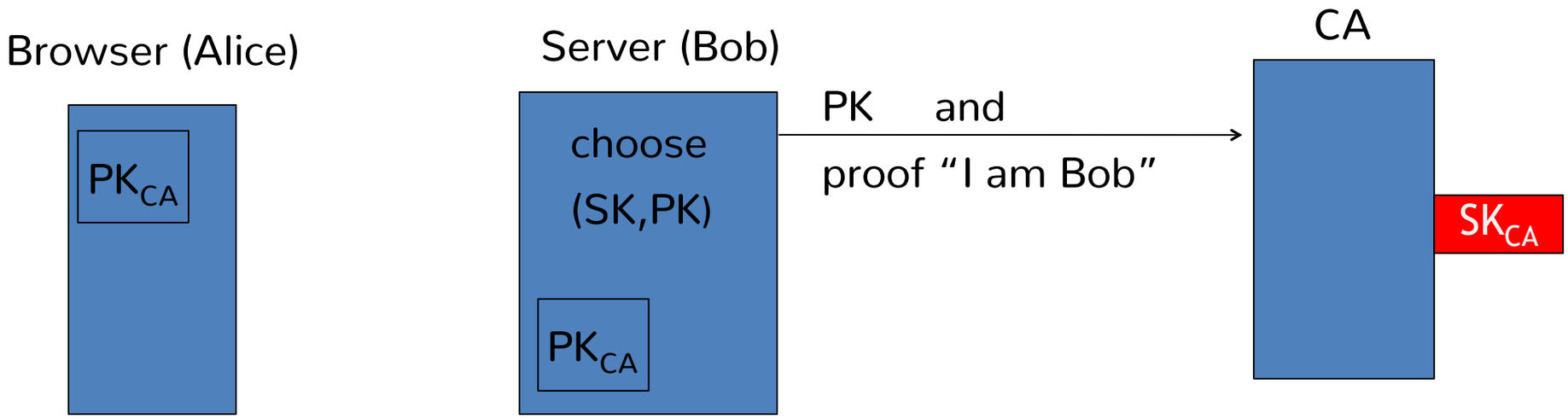
Certificate Authorities

- **Certificate Authority:** Trusted authority
  - Signs keys after checking some kind of identity (e.g., you own www.google.com)
  - Server presents signed key (cert) to clients
  - Browsers and OSes ship with public keys of trusted CAs

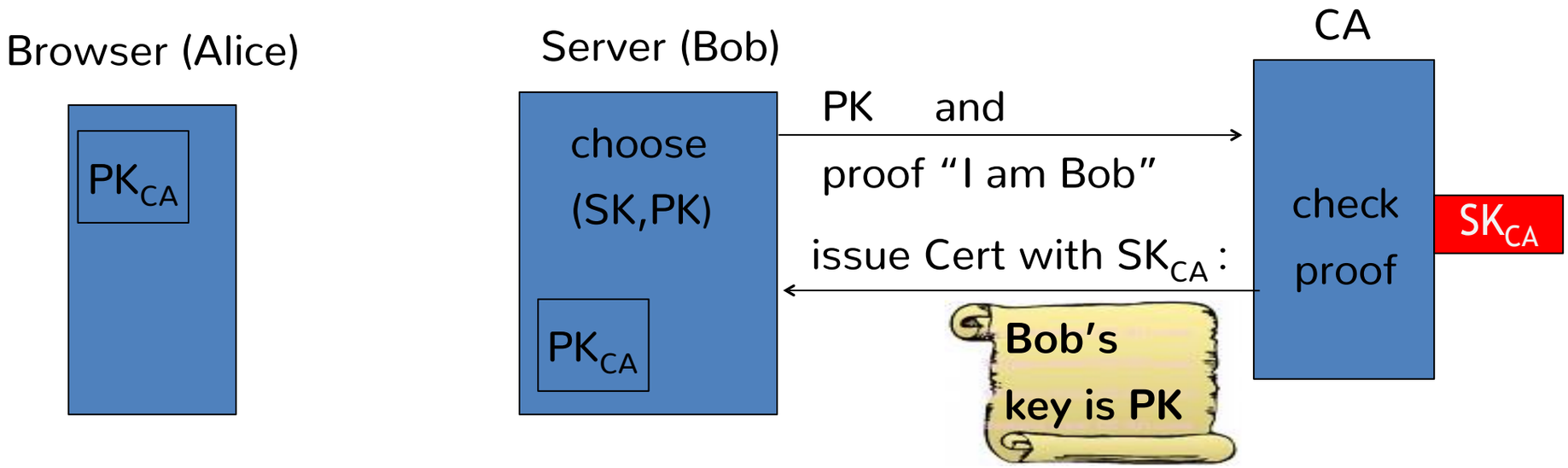
Certificate Authorities



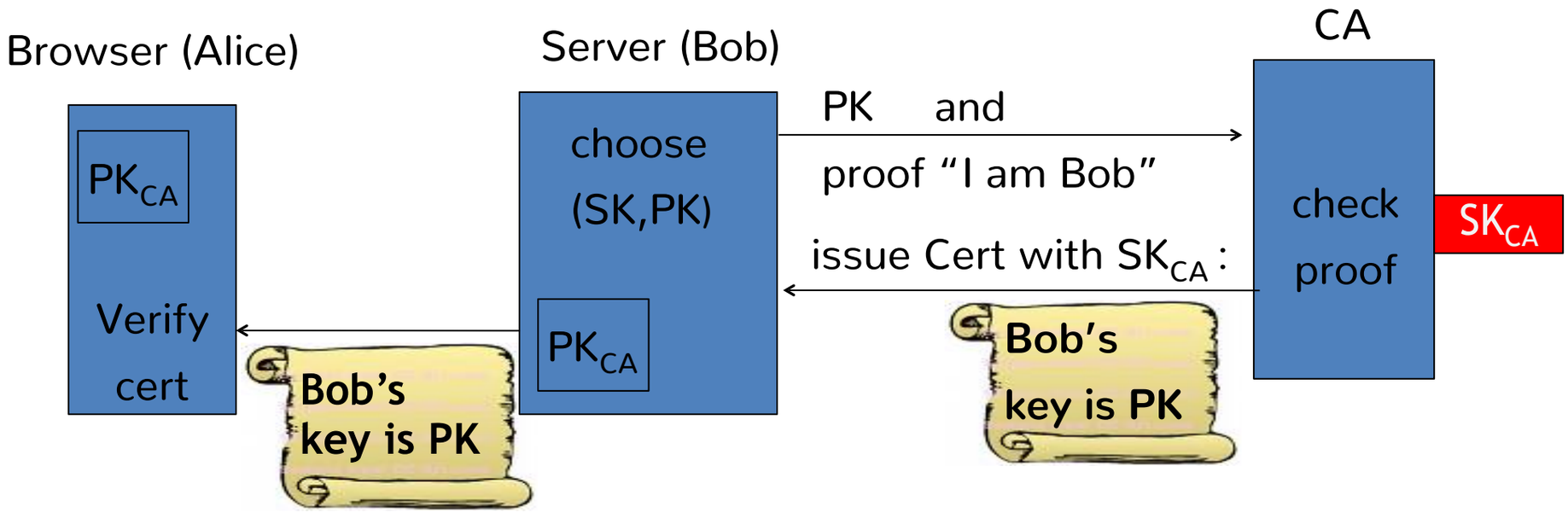
Certificate Authorities



Certificate Authorities



Certificate Authorities

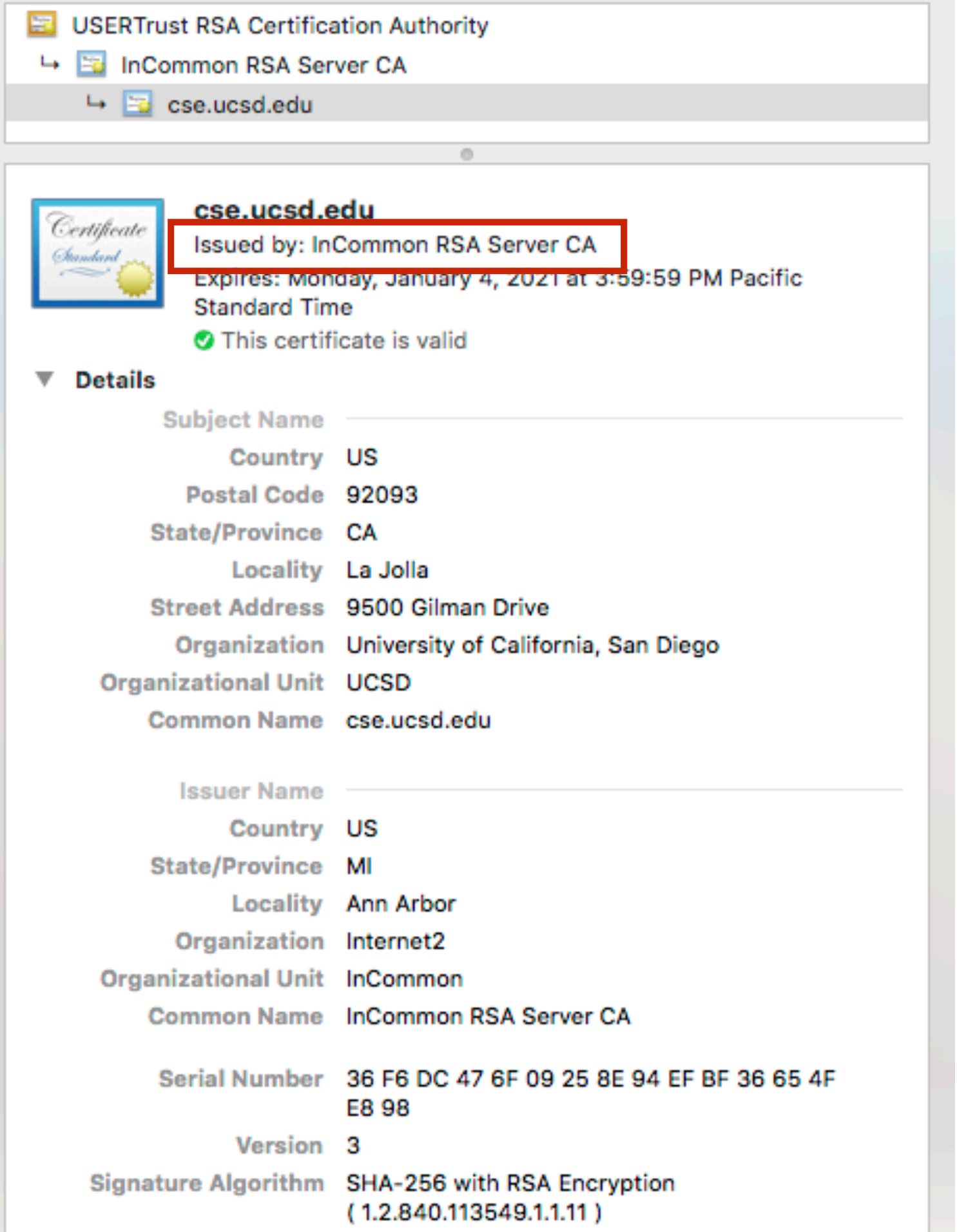


USERTrust RSA Certification Authority
↳ InCommon RSA Server CA
↳ cse.ucsd.edu

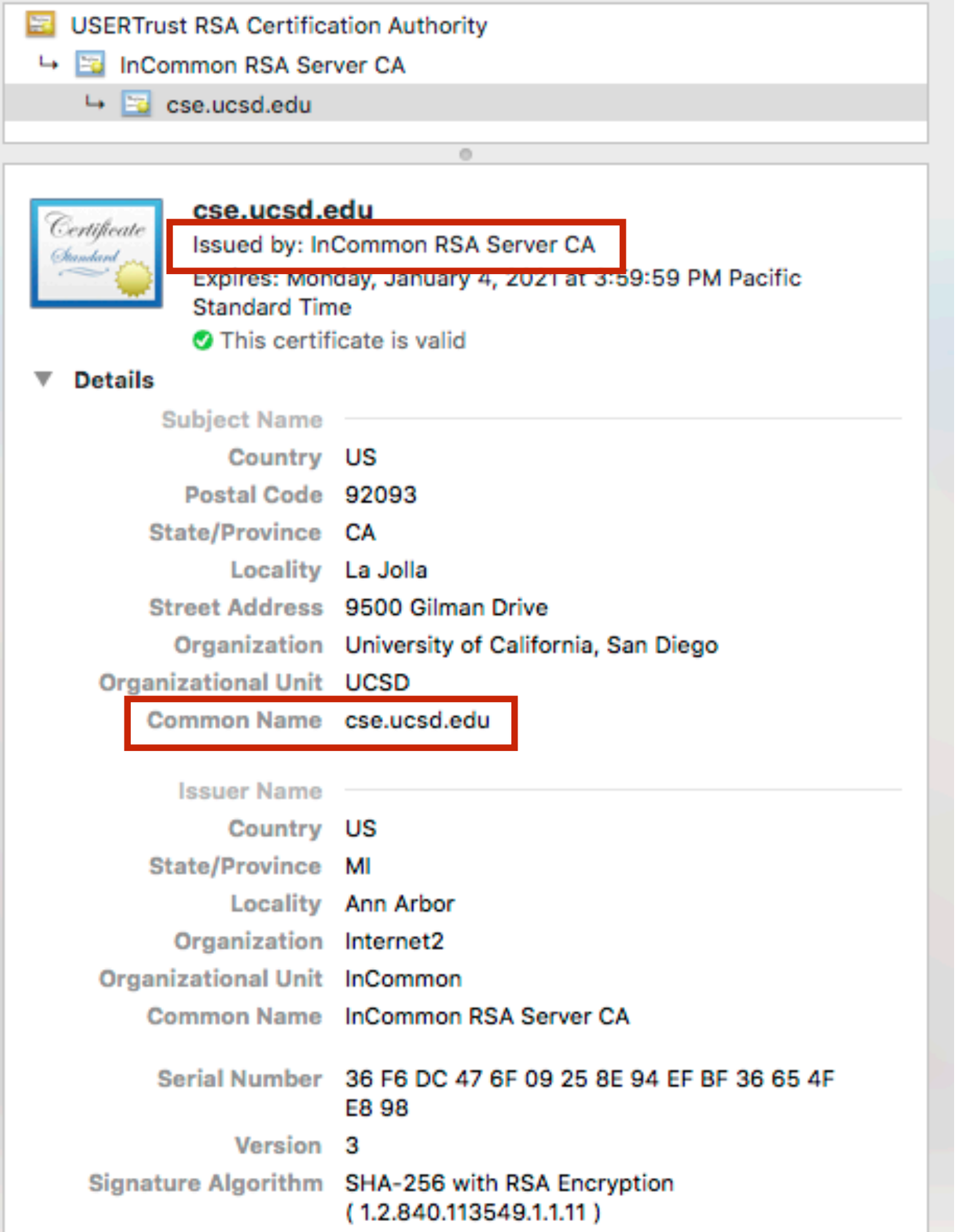
cse.ucsd.edu
Issued by: InCommon RSA Server CA
Expires: Monday, January 4, 2021 at 3:59:59 PM Pacific Standard Time
This certificate is valid

Details

Subject Name	
Country	US
Postal Code	92093
State/Province	CA
Locality	La Jolla
Street Address	9500 Gilman Drive
Organization	University of California, San Diego
Organizational Unit	UCSD
Common Name	cse.ucsd.edu
Issuer Name	
Country	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA
Serial Number	36 F6 DC 47 6F 09 25 8E 94 EF BF 36 65 4F E8 98
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)



Who are we trusting?



Who are we trusting?

Who is this cert for?

Key ID	1E 05 A3 77 8F 6C 96 E2 5B 87 4B A6 B4 86 AC 71 00 0C E7 38
Extension	Subject Alternative Name (2.5.29.17)
Critical	NO
DNS Name	cse.ucsd.edu
DNS Name	cs.ucsd.edu
DNS Name	www-cs.ucsd.edu
DNS Name	www-cse.ucsd.edu
DNS Name	www.cs.ucsd.edu
DNS Name	www.cse.ucsd.edu
Extension	Certificate Policies (2.5.29.32)
Critical	NO
Policy ID #1	(1.3.6.1.4.1.5923.1.4.3.1.1)
Qualifier ID #1	Certification Practice Statement (1.3.6.1.5.5.7.2.1)
CPS URI	https://www.incommon.org/cert/repository/cps_ssl.pdf
Policy ID #2	(2.23.140.1.2.2)
Extension	CRL Distribution Points (2.5.29.31)
Critical	NO
URI	http://crl.incommon-rsa.org/InCommonRSAServerCA.crl
Extension	Certificate Authority Information Access (1.3.6.1.5.5.7.1.1)
Critical	NO
Method #1	CA Issuers (1.3.6.1.5.5.7.48.2)
URI	http://crt.usertrust.com/InCommonRSAServerCA_2.crt
Method #2	Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)
URI	http://ocsp.usertrust.com

Key ID	1E 05 A3 77 8F 6C 96 E2 5B 87 4B A6 B4 86 AC 71 00 0C E7 38
Extension	Subject Alternative Name (2.5.29.17)
Critical	NO
DNS Name	cse.ucsd.edu
DNS Name	cs.ucsd.edu
DNS Name	www-cs.ucsd.edu
DNS Name	www-cse.ucsd.edu
DNS Name	www.cs.ucsd.edu
DNS Name	www.cse.ucsd.edu
Extension	Certificate Policies (2.5.29.32)
Critical	NO
Policy ID #1	(1.3.6.1.4.1.5923.1.4.3.1.1)
Qualifier ID #1	Certification Practice Statement (1.3.6.1.5.5.7.2.1)
CPS URI	https://www.incommon.org/cert/repository/cps_ssl.pdf
Policy ID #2	(2.23.140.1.2.2)
Extension	CRL Distribution Points (2.5.29.31)
Critical	NO
URI	http://crl.incommon-rsa.org/InCommonRSAServerCA.crl
Extension	Certificate Authority Information Access (1.3.6.1.5.5.7.1.1)
Critical	NO
Method #1	CA Issuers (1.3.6.1.5.5.7.48.2)
URI	http://crt.usertrust.com/InCommonRSAServerCA_2.crt
Method #2	Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)
URI	http://ocsp.usertrust.com

Who is this cert for?

Issuer Name	
Country	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA
Serial Number	36 F6 DC 47 6F 09 25 8E 94 EF BF 36 65 4F E8 98
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	None
Not Valid Before	Thursday, January 4, 2018 at 4:00:00 PM Pacific Standard Time
Not Valid After	Monday, January 4, 2021 at 3:59:59 PM Pacific Standard Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes : FA F9 1A 08 92 86 9C 7B ...
Exponent	65537
Key Size	2,048 bits
Key Usage	Encrypt, Verify, Wrap, Derive
Signature	256 bytes : 6F 62 36 46 B7 43 28 04 ...
Extension	Key Usage (2.5.29.15)
Critical	YES
Usage	Digital Signature, Key Encipherment

Issuer Name	
Country	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA
Serial Number	36 F6 DC 47 6F 09 25 8E 94 EF BF 36 65 4F E8 98
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	None
Not Valid Before	Thursday, January 4, 2018 at 4:00:00 PM Pacific Standard Time
Not Valid After	Monday, January 4, 2021 at 3:59:59 PM Pacific Standard Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes : FA F9 1A 08 92 86 9C 7B ...
Exponent	65537
Key Size	2,048 bits
Key Usage	Encrypt, Verify, Wrap, Derive
Signature	256 bytes : 6F 62 36 46 B7 43 28 04 ...
Extension	Key Usage (2.5.29.15)
Critical	YES
Usage	Digital Signature, Key Encipherment

CSE's pub key info

Key ID	1E 05 A3 77 8F 6C 96 E2 5B 87 4B A6 B4 86 AC 71 00 0C E7 38
Extension	Subject Alternative Name (2.5.29.17)
Critical	NO
DNS Name	cse.ucsd.edu
DNS Name	cs.ucsd.edu
DNS Name	www-cs.ucsd.edu
DNS Name	www-cse.ucsd.edu
DNS Name	www.cs.ucsd.edu
DNS Name	www.cse.ucsd.edu
Extension	Certificate Policies (2.5.29.32)
Critical	NO
Policy ID #1	(1.3.6.1.4.1.5923.1.4.3.1.1)
Qualifier ID #1	Certification Practice Statement (1.3.6.1.5.5.7.2.1)
CPS URI	https://www.incommon.org/cert/repository/cps_ssl.pdf
Policy ID #2	(2.23.140.1.2.2)
Extension	CRL Distribution Points (2.5.29.31)
Critical	NO
URI	http://crl.incommon-rsa.org/InCommonRSAServerCA.crl
Extension	Certificate Authority Information Access (1.3.6.1.5.5.7.1.1)
Critical	NO
Method #1	CA Issuers (1.3.6.1.5.5.7.48.2)
URI	http://crt.usertrust.com/InCommonRSAServerCA_2.crt
Method #2	Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)
URI	http://ocsp.usertrust.com

Key ID	1E 05 A3 77 8F 6C 96 E2 5B 87 4B A6 B4 86 AC 71 00 0C E7 38
Extension	Subject Alternative Name (2.5.29.17)
Critical	NO
DNS Name	cse.ucsd.edu
DNS Name	cs.ucsd.edu
DNS Name	www-cs.ucsd.edu
DNS Name	www-cse.ucsd.edu
DNS Name	www.cs.ucsd.edu
DNS Name	www.cse.ucsd.edu
Extension	Certificate Policies (2.5.29.32)
Critical	NO
Policy ID #1	(1.3.6.1.4.1.5923.1.4.3.1.1)
Qualifier ID #1	Certification Practice Statement (1.3.6.1.5.5.7.2.1)
CPS URI	https://www.incommon.org/cert/repository/cps_ssl.pdf
Policy ID #2	(2.23.140.1.2.2)
Extension	CRL Distribution Points (2.5.29.31)
Critical	NO
URI	http://crl.incommon-rsa.org/InCommonRSAServerCA.crl
Extension	Certificate Authority Information Access (1.3.6.1.5.5.7.1.1)
Critical	NO
Method #1	CA Issuers (1.3.6.1.5.5.7.48.2)
URI	http://crt.usertrust.com/InCommonRSAServerCA_2.crt
Method #2	Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)
URI	http://ocsp.usertrust.com

Key ID	1E 05 A3 77 8F 6C 96 E2 5B 87 4B A6 B4 86 AC 71 00 0C E7 38
Extension	Subject Alternative Name (2.5.29.17)
Critical	NO
DNS Name	cse.ucsd.edu
DNS Name	cs.ucsd.edu
DNS Name	www-cs.ucsd.edu
DNS Name	www-cse.ucsd.edu
DNS Name	www.cs.ucsd.edu
DNS Name	www.cse.ucsd.edu
Extension	Certificate Policies (2.5.29.32)
Critical	NO
Policy ID #1	(1.3.6.1.4.1.5923.1.4.3.1.1)
Qualifier ID #1	Certification Practice Statement (1.3.6.1.5.5.7.2.1)
CPS URI	https://www.incommon.org/cert/repository/cps_ssl.pdf
Policy ID #2	(2.23.140.1.2.2)
Extension	CRL Distribution Points (2.5.29.31)
Critical	NO
URI	http://crl.incommon-rsa.org/InCommonRSAServerCA.crl
Extension	Certificate Authority Information Access (1.3.6.1.5.5.7.1.1)
Critical	NO
Method #1	CA Issuers (1.3.6.1.5.5.7.48.2)
URI	http://crt.usertrust.com/InCommonRSAServerCA_2.crt
Method #2	Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)
URI	http://ocsp.usertrust.com

Where we should check for revocation information

Revocation

- Problem: keys get compromised
  - Attacker with a key can impersonate you and read messages encrypted to you
- Key expiration helps with this but not enough
- CA and PGP PKIs support revocation
  - "I, Alice, revoke my public key … do not use it."
  - Signs revocation with her private key
  - Others can verify Alice's signature, stop using key

Revocation

- In CA model, Alice asks CA to revoke certificate
  - Alice does not need private key to do this
  - CAs publish a Certificate Revocation List (CRL)
- In PGP model, only Alice can revoke her own key
  - If Alice loses her private key, she can't revoke
  - Do not lose private key
  - Option: generate revocation with key, store in secure place

Revocation

- CRL: Certificate Revocation List
  - CA publishes list of revoked certs
  - Client downloads the list
  - Problem: Only care about few certs.
  - Problem: What if CRL server is down?

Revocation

- OCSP: Online Certificate Status Protocol
  - Query CA about cert when you get it from server
  - Problem: Revealing visited sites to CA.
- OCSP stapling: Web server includes recent OCSP cert

Sometimes CAs go wrong

- CAs get hacked or do the wrong thing:
  - 2011: Comodo and DigiNotar CAs hacked, issue certs for Gmail, Yahoo! Mail, …
  - 2013: TurkTrust issued cert for Gmail
  - 2014: Indian NIC issue certs for Google and Yahoo!
  - 2016: WoSignissues cert for GitHub
- Solution: Certificate transparency!
  - CAs have to publish certs they issue

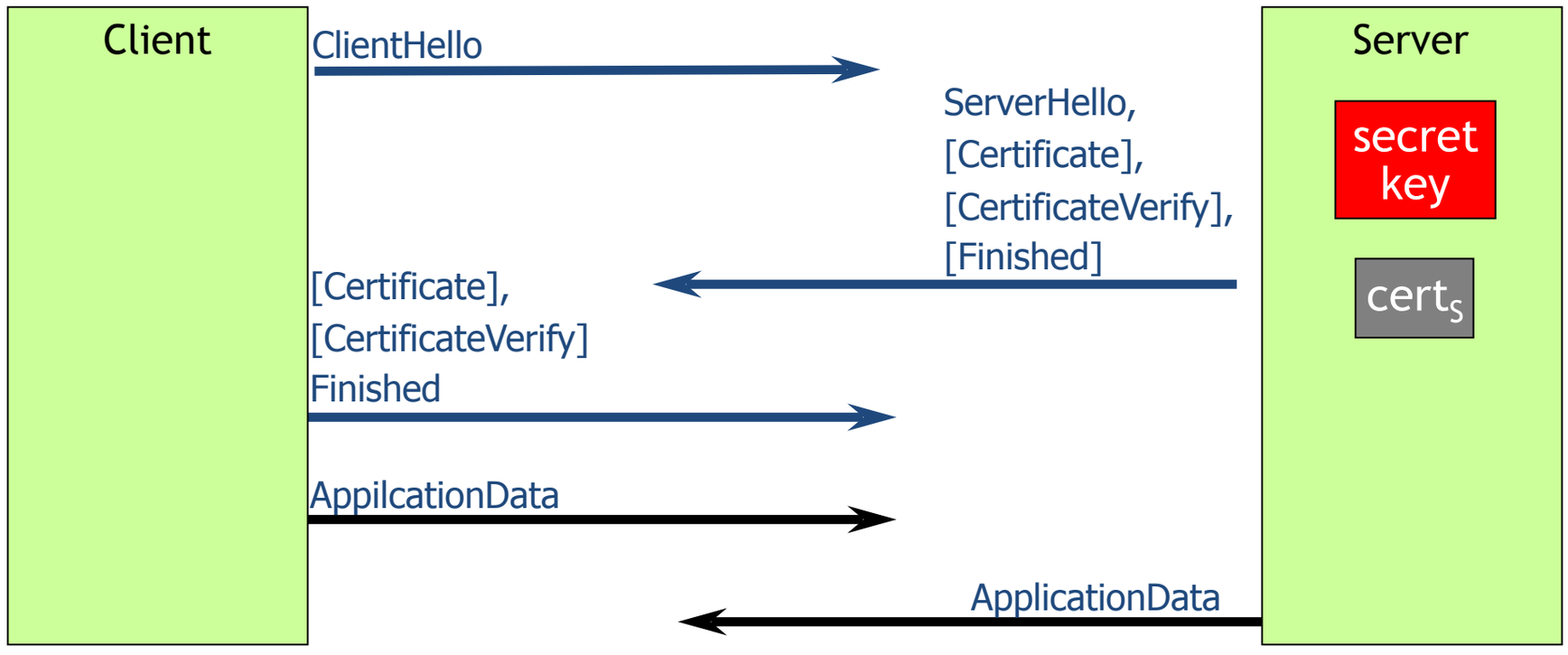
crt.sh Mozilla CA Certificate Disclosures

Generated at 2019-03-14 00:49:38 UTC

Category	Disclosure Required?	# of CA certs
Disclosure Incomplete	**Yes!**	1 + 6 Summary
Unconstrained Trust	**Yes!**	1 + 15 Summary
Unconstrained, but all unexpired observed paths Revoked	Unknown	46
Unconstrained, but zero unexpired observed paths	Unknown	1640
Expired	No	4437
Technically Constrained (Trusted)	Maybe soon?	56
Technically Constrained (Other)	No	40
Disclosed as Revoked, but Expired	Already disclosed	156
Disclosed as Revoked and in OneCRL	Already disclosed	575
Disclosed as Revoked (but not in OneCRL)	Already disclosed	32
Disclosed as Parent Revoked (so not in OneCRL)	Already disclosed	142
Disclosed, but Expired	Already disclosed	466
Disclosed, but zero unexpired observed paths	Already disclosed	496
Disclosed (as Not Revoked), but in OneCRL	Already disclosed	28
Disclosed, but Technically Constrained	Already disclosed	240
Disclosed, but with Errors	Already disclosed	0
Disclosed (as Not Revoked), but Revoked via CRL	Already disclosed	9
Disclosed (as Not Revoked) and "Unrevoked" from CRL	Already disclosed	2
Disclosed	Already disclosed	3170
Unknown to crt.sh or Incorrectly Encoded	Already disclosed	6

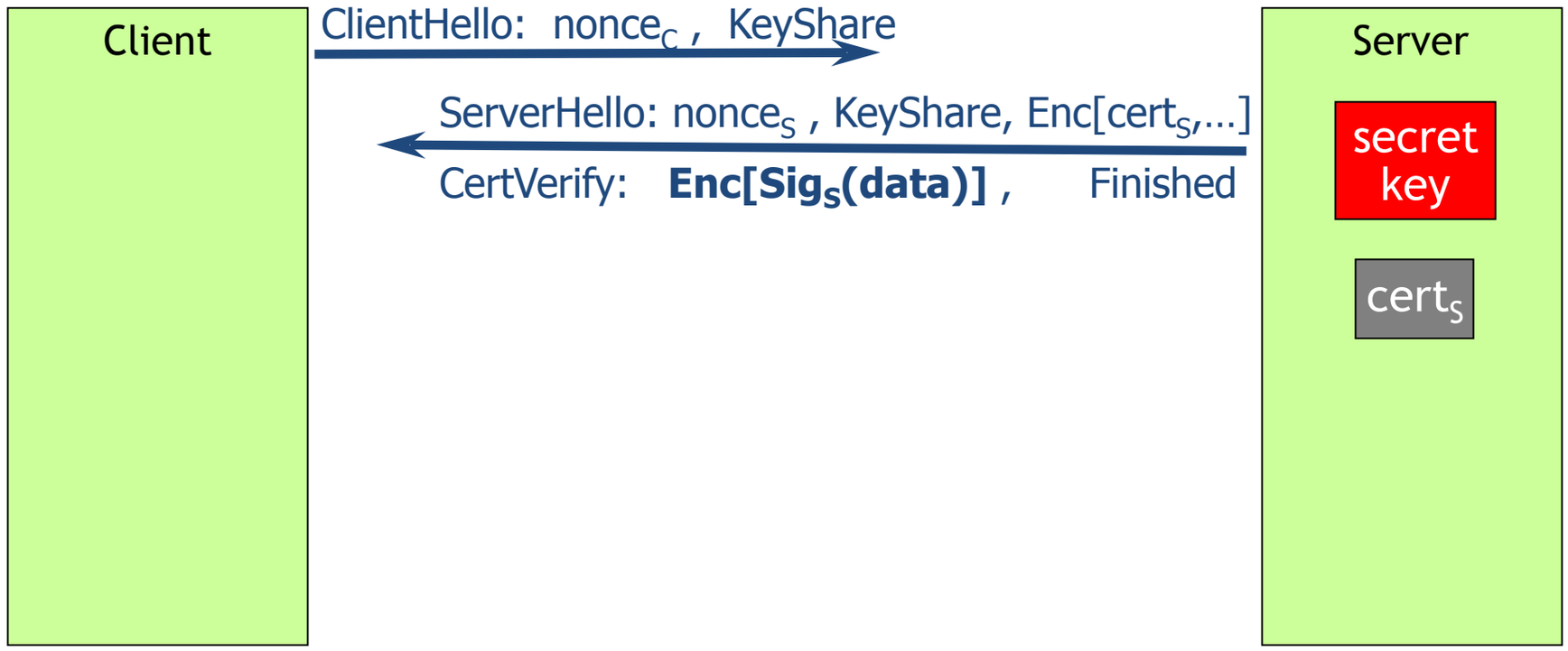
How is asymmetric key crypto used?

How are these used in TLS 1.3?

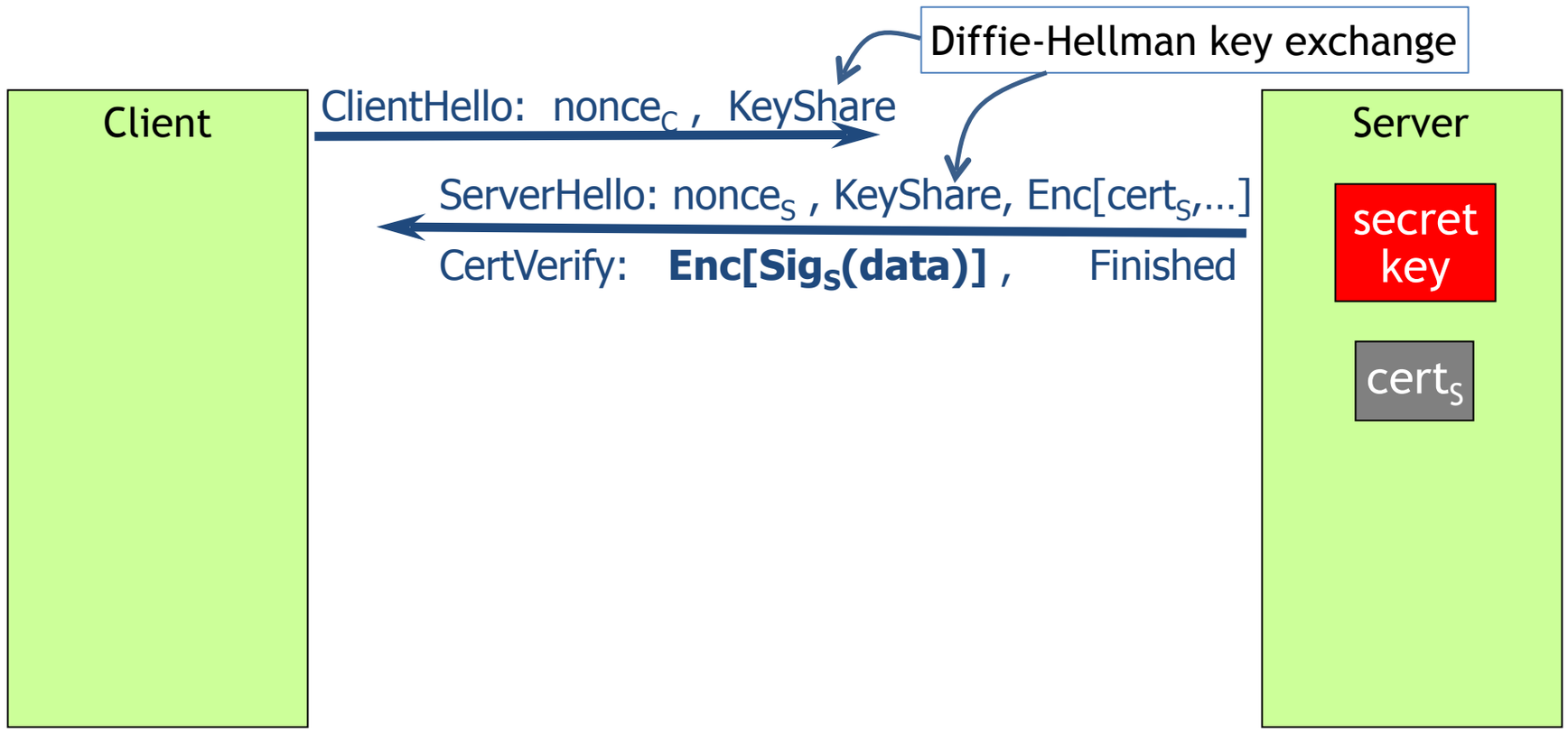


How are these used in TLS 1.3?

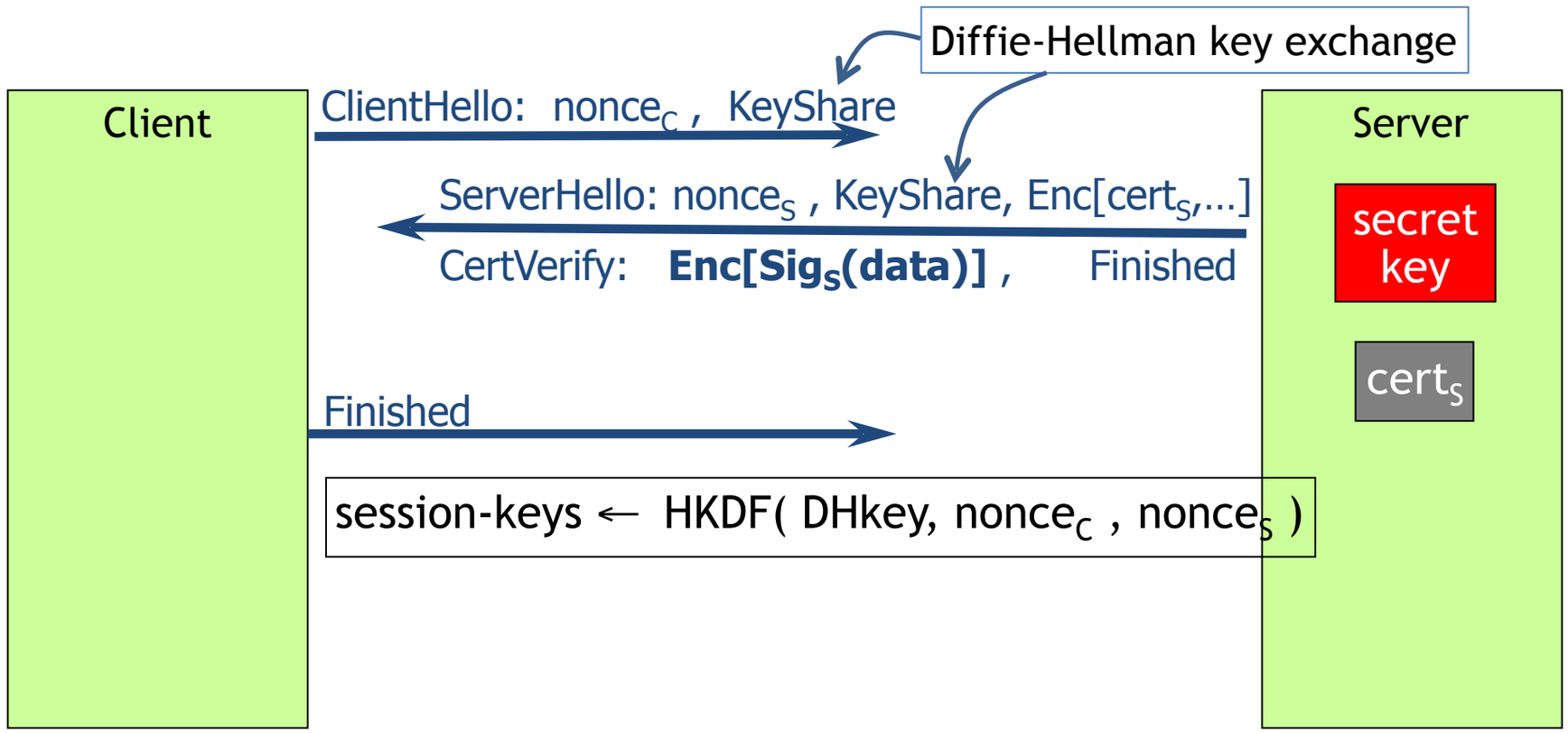
How are these used in TLS 1.3?



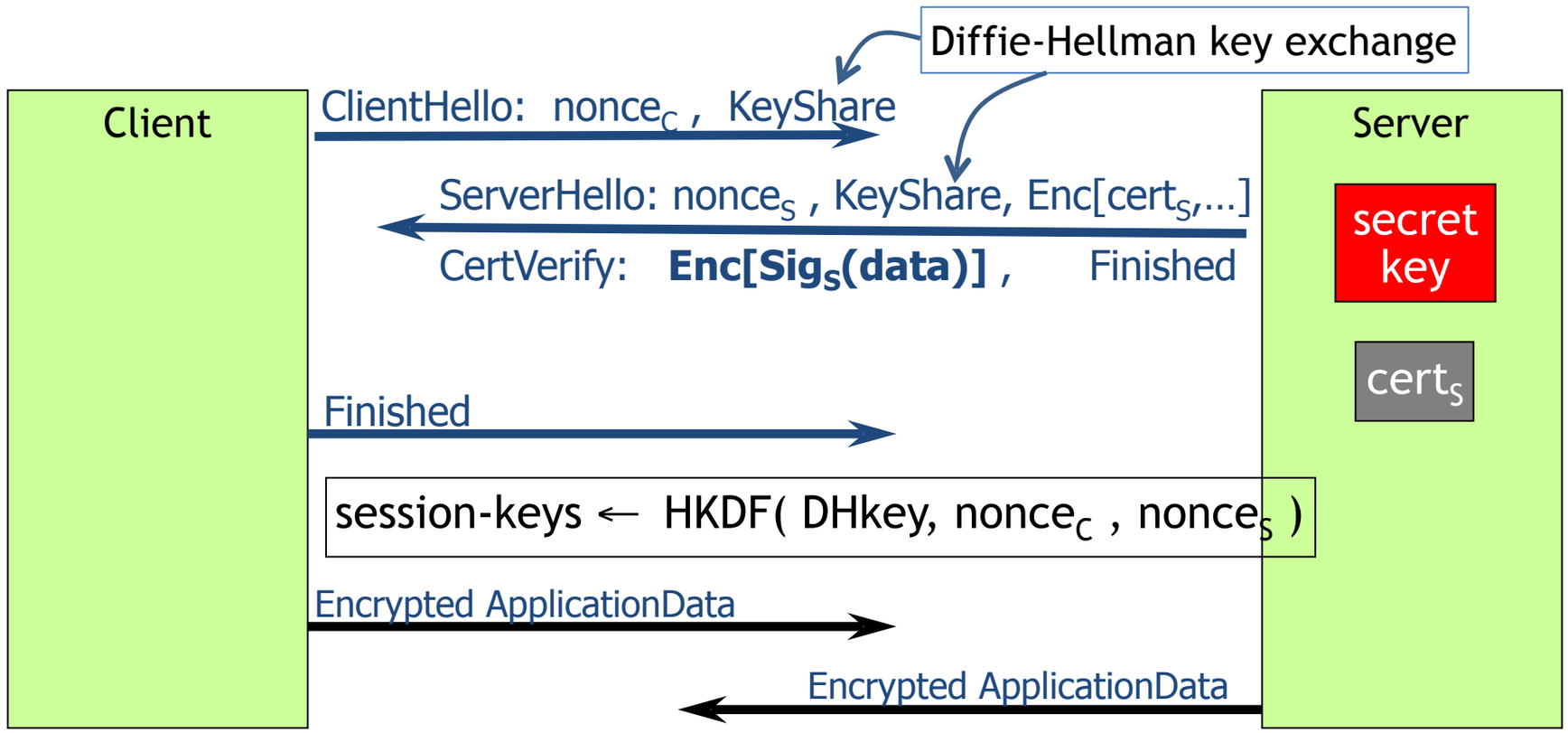
How are these used in TLS 1.3?



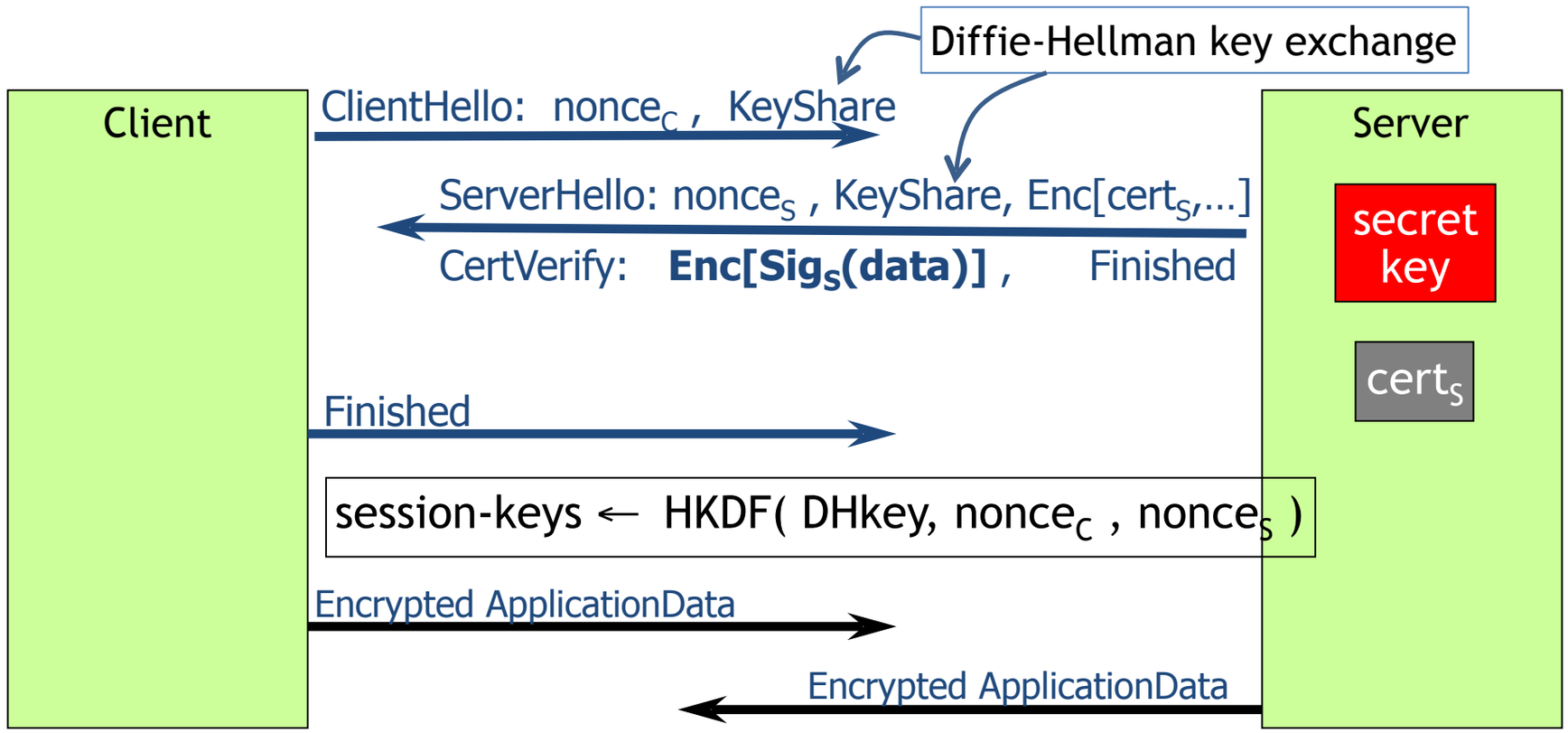
How are these used in TLS 1.3?



How are these used in TLS 1.3?



How are these used in TLS 1.3?

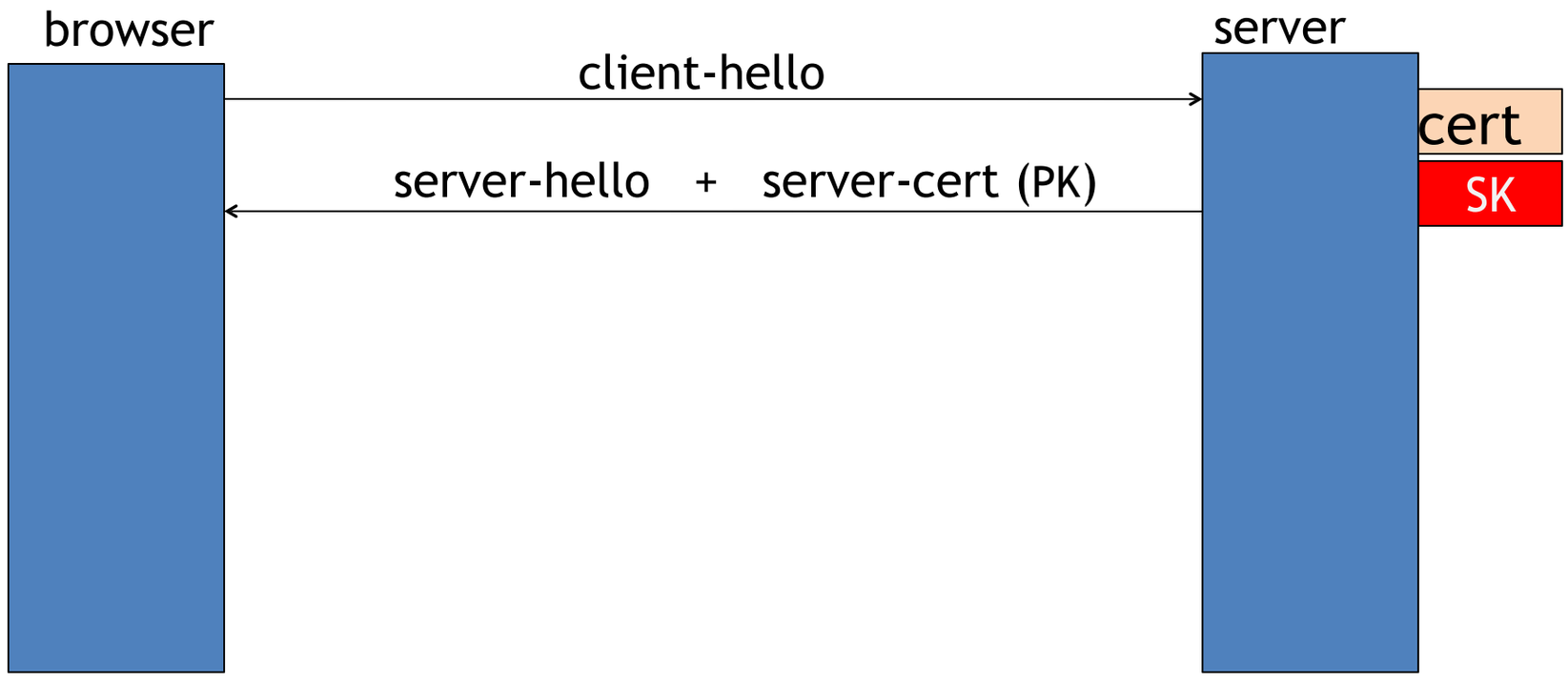


TLS 1.2 handshake is way longer, requires encrypting w/ server public key, etc.

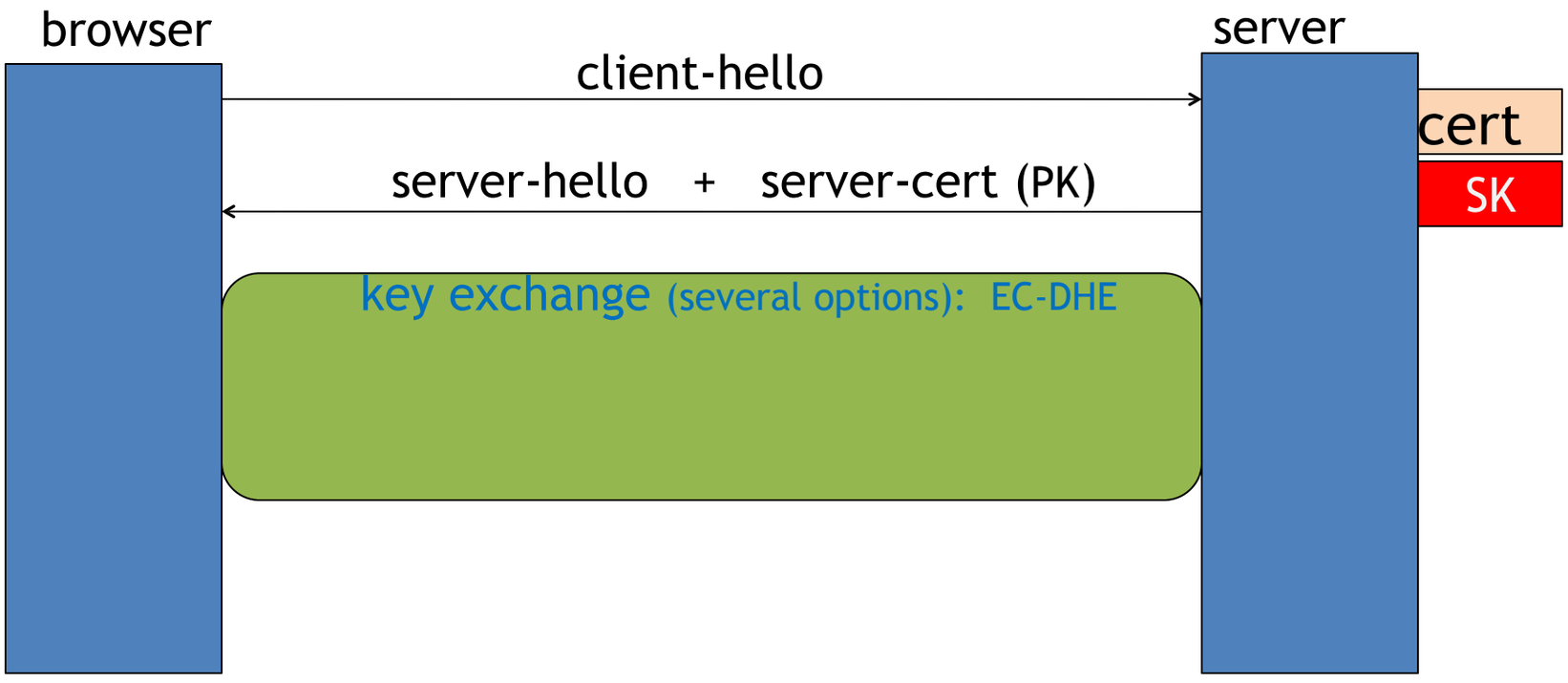
Brief overview of TLS 1.2

Brief overview of TLS 1.2

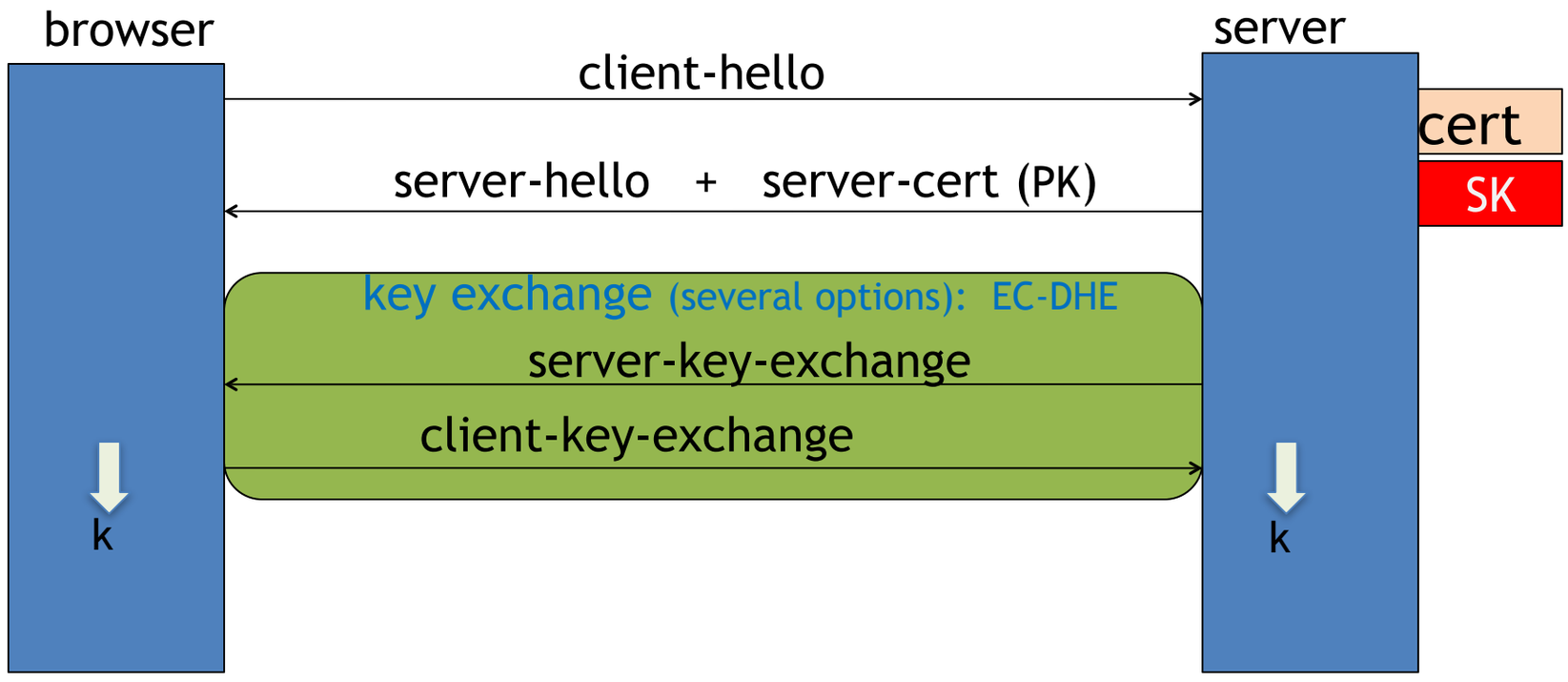
Brief overview of TLS 1.2



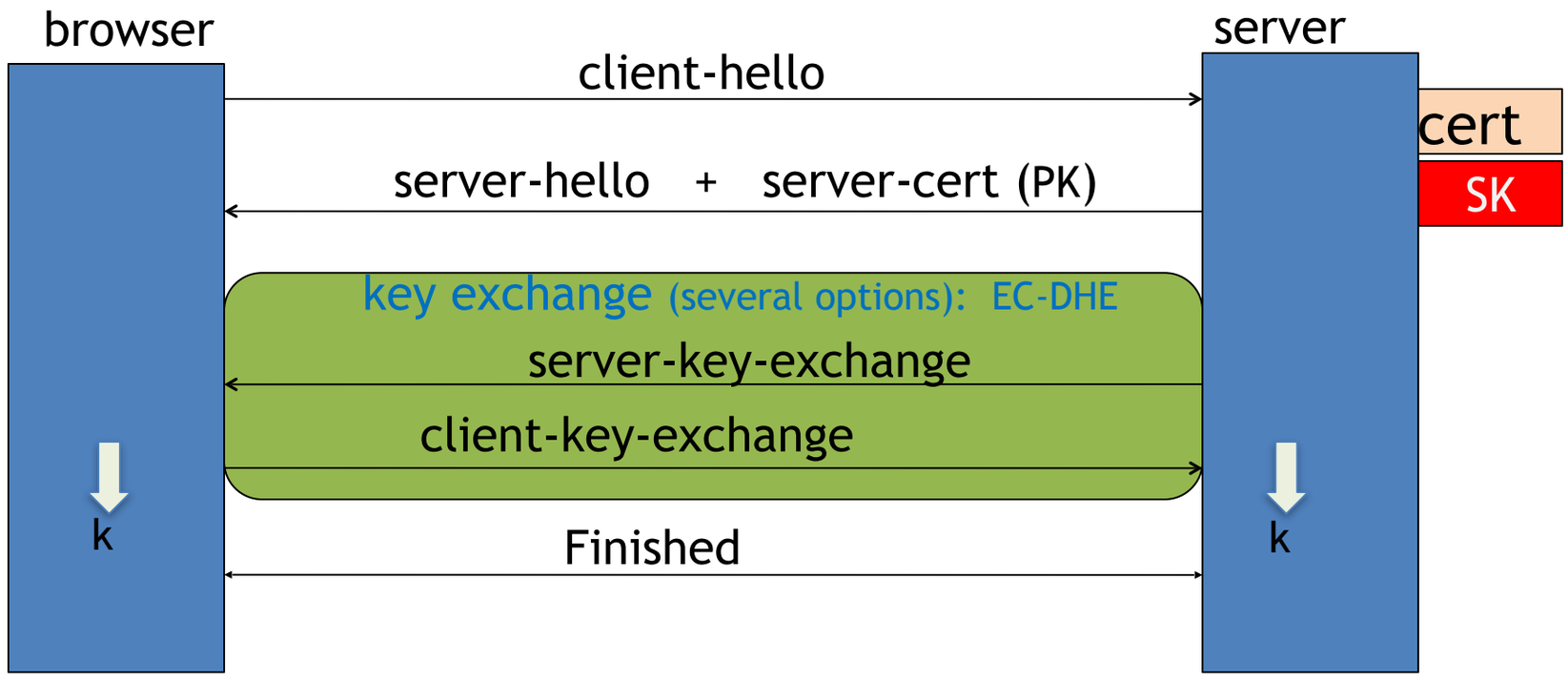
Brief overview of TLS 1.2



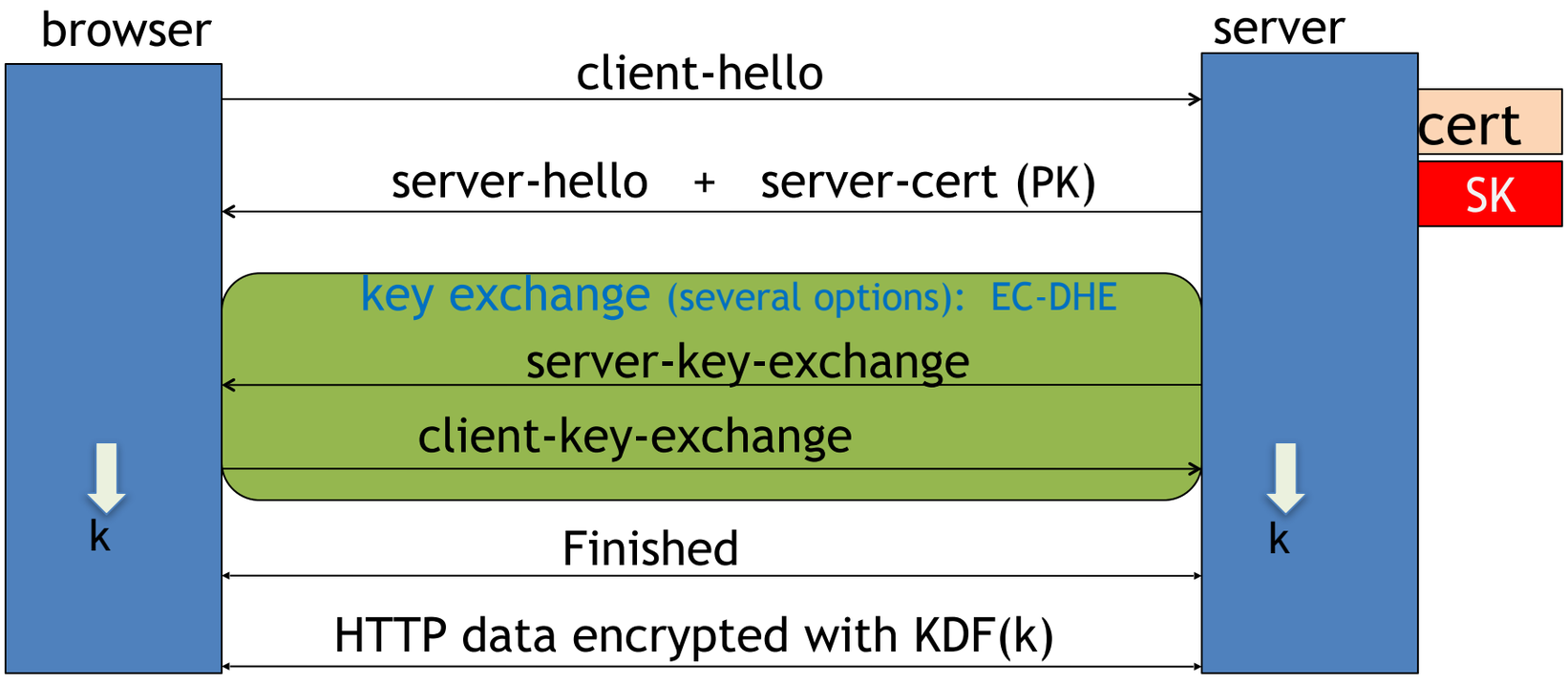
Brief overview of TLS 1.2



Brief overview of TLS 1.2



Brief overview of TLS 1.2



What does TLS give you?

- **Nonces:** prevent replay of an old session
- **Forward secrecy:** server compromise does not expose old sessions
- **Some identity protection:** certificates are sent encrypted
  - SNI also encrypted so can't see which server in clear
- **(One sided) authentication:** browser identifies server using server-cert

No excuse not use TLS!