Cryptography

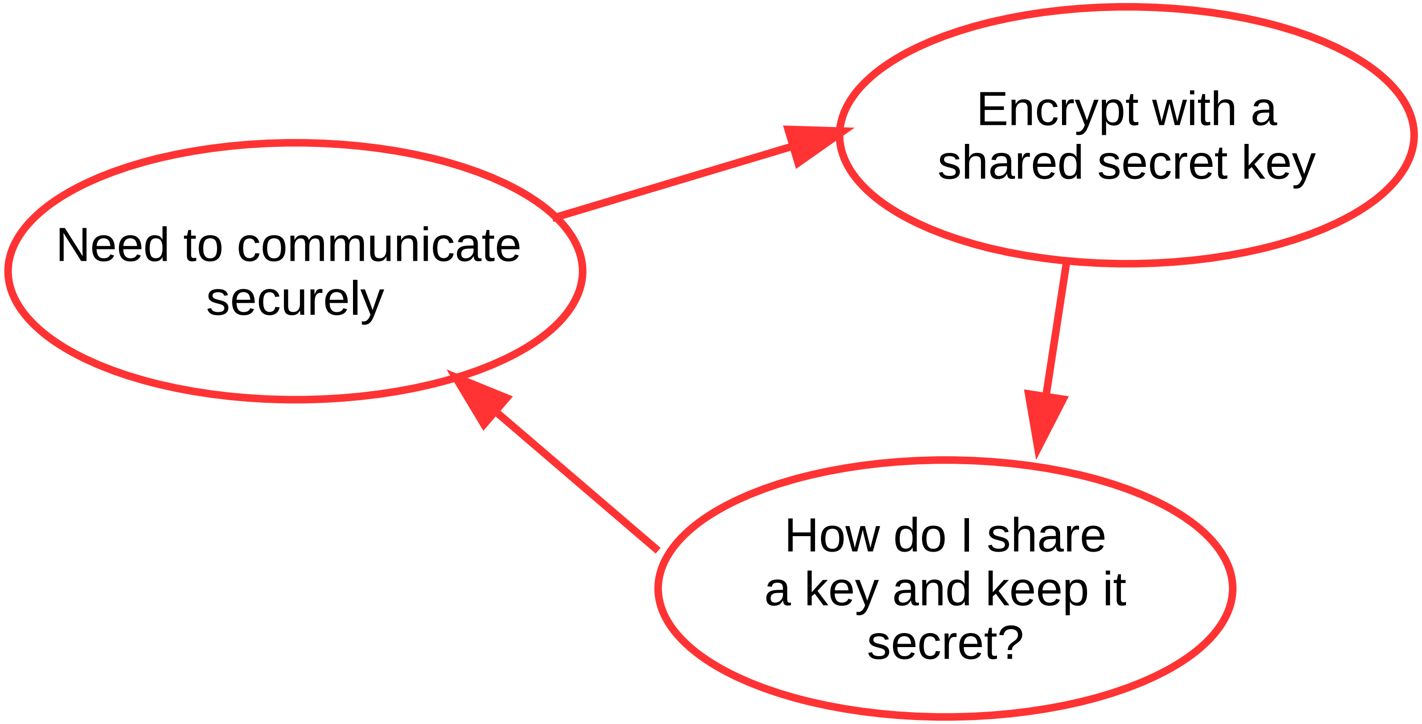
public-key cryptography

Key management

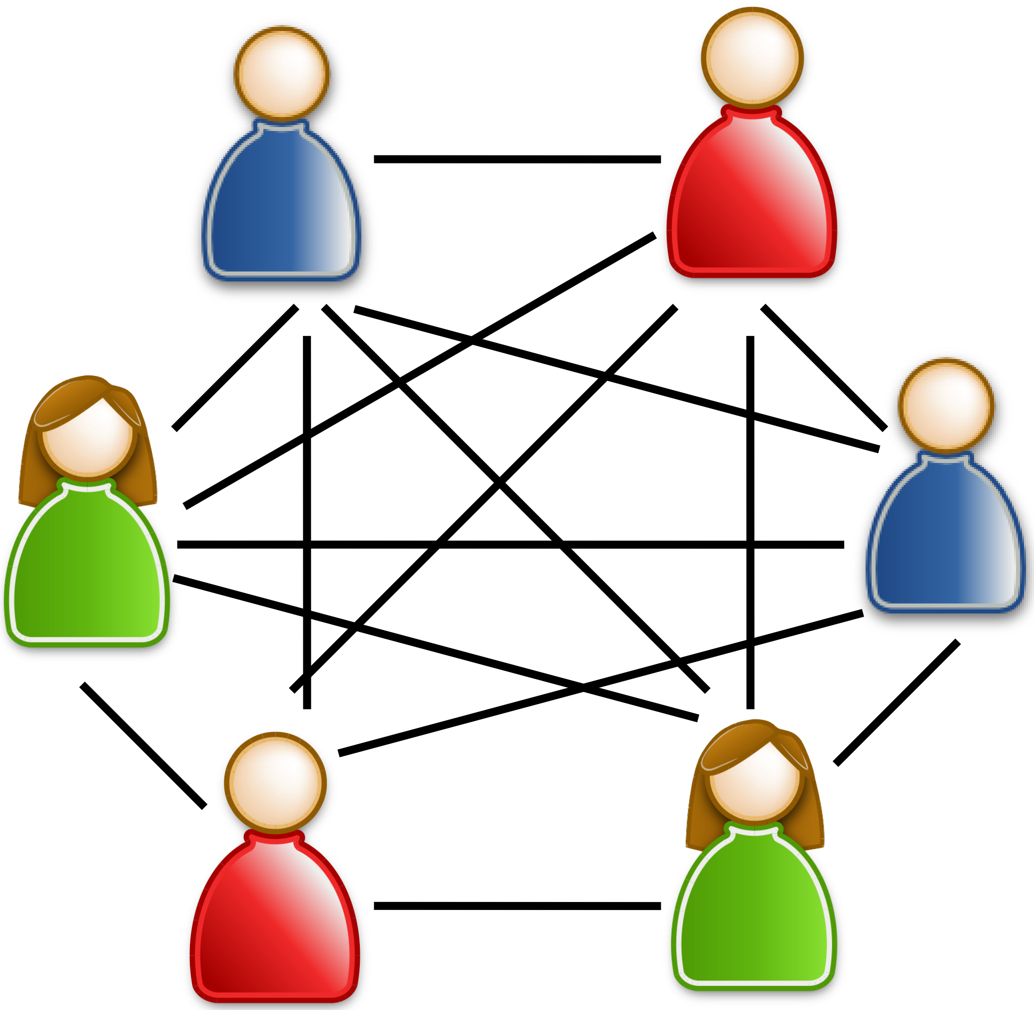
part 1

Problem

We can send encrypted documents - to people who have a key. We've turned a security problem into a *key management problem*.



Problem



n people =
O(n²) keys needed

You also have to update keys every now and then.

Does this look like a mess to you?

What if you want to chat to some-
one you've never met before?

The padlock analogy

Imagine a padlock that you can just “click” shut when it’s open.

To open it when it’s shut, you need the key

I open my padlock and give it to you. You can write me a message and lock it in a box.

You can send me a locked box without ever seeing my key.

public-key encryption

The padlock analogy

Public-key cryptography uses two kinds of keys:

public keys are like padlocks - you can hand them out to anyone, with a public key you can only lock but not unlock things.

secret keys (also called private keys) are like … keys. You use them to unlock things, and you never give them out to others.

Public-key cryptography

Every user has a key pair:

sk secret key

known only to the
user themselves

pk public key

published in a
directory / website

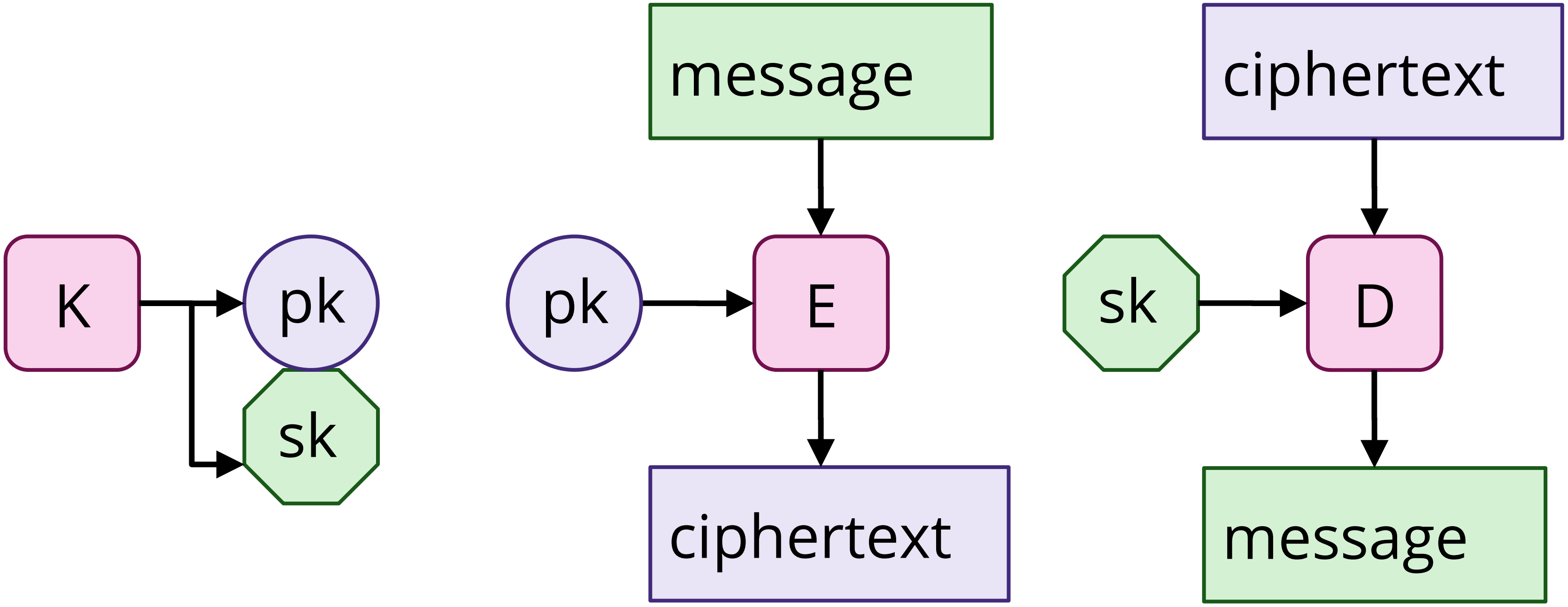
usually:

pk = f(sk)

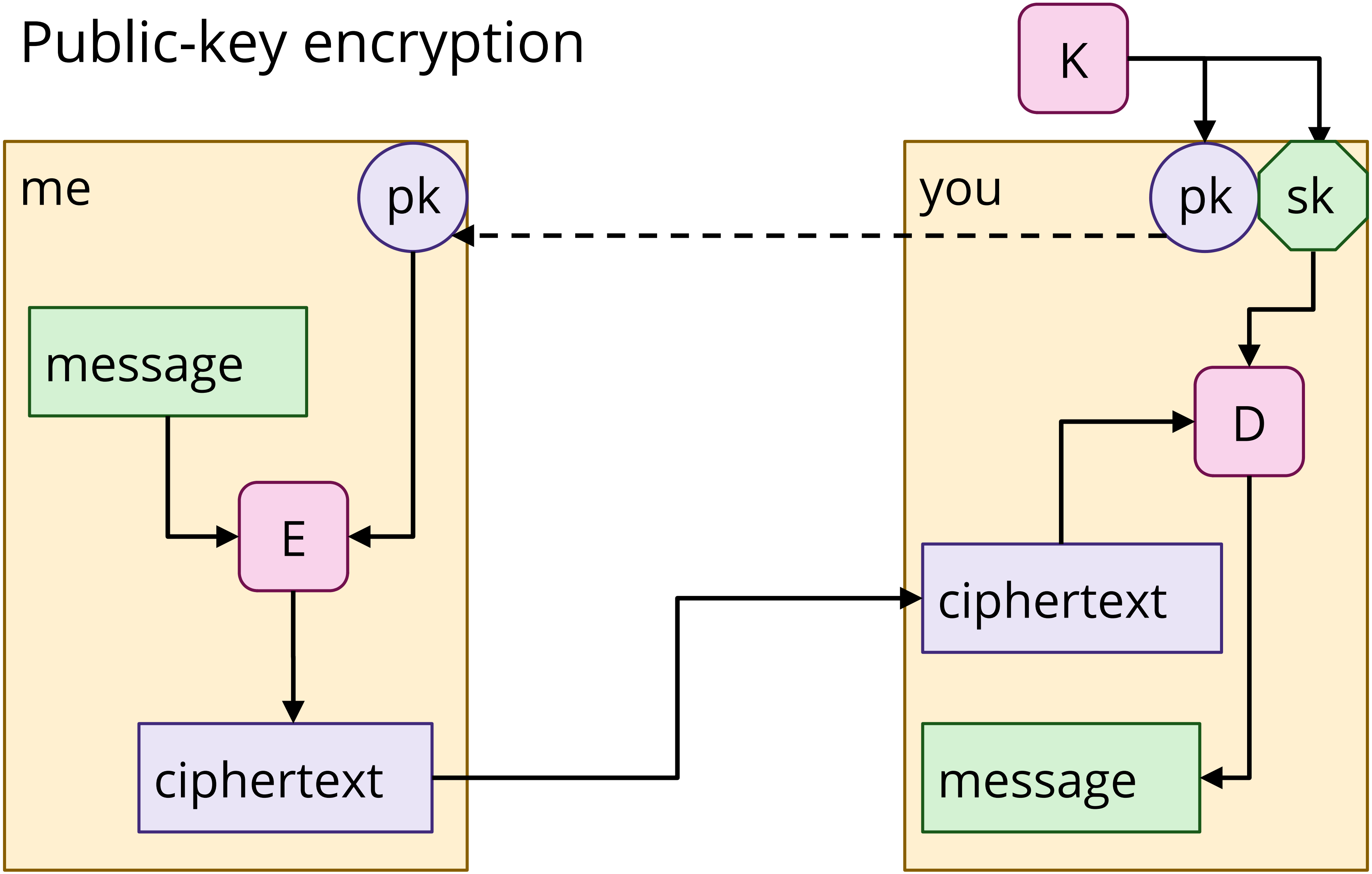
Usually, if you have a secret key then it is easy to compute the matching public key.

But the other way round is (for all practical purposes) impossible.

Public-key encryption



Public-key encryption



public-key encryption

A public-key encryption scheme contains three algorithms:

The key generation algorithm **K** creates a keypair (pk, sk) containing a public and a secret key.

The encryption algorithm **E** takes a public key and a message and returns a ciphertext.

The decryption algorithm **D** takes a secret key and a ciphertext and returns a message, or an error.

If **(pk, sk)** comes from **K** then **D(sk, E(pk, m)) = m**.

Encrypted e-mail

From: [Sender]
Subject: encrypted e-mail

Message encrypted under your PK.

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1
```


```
hQIOA9+KD3n+20B6EAf+P5mIK327bkk0rs86ecFgG1gOnGP37RZD6
PMzv0QMaLm0QTc5vmc+L1BJ7SXZo+fM4JrpL07k0qD28JbJsv/cQK
2Qzhs3RiE3KTCcJJqNizF1yLfpJ3rJqSnZmcpI5LWEENpQyd9Fbe8
X1mxjrAHqIQBfhVG8xmAqv+VW4grJORA5UrRAwXRoGwM5jpvR+Jqd
cLiWd6Et9MD5Ef7SfwOhvVddTIxY2ed4mXucS7AfbbPR503Mp38ro
=B6gF
```


```
-----END PGP MESSAGE-----
```

Edward Snowden

2013: Edward Snowden, NSA contractor, flies to Hong Kong and leaks secret documents to Glenn Greenwald (journalist) at the Guardian and other papers.

He is currently living in Russia, seeking asylum.

He sent the files to the Guardian using GPG (and wrote a tutorial) - he claims that if done properly, even the government/NSA can't break its encryption.

RSA

RSA (outline)

1. Choose any two distinct prime numbers, *p* & *q*
2. Compute *n* = *pq*
3. Select a prime number *e* between 1 and *lcm*(*p*-1,*q*-1)
4. The **public key** is <n, e>
5. Compute the number *d* such that
   (*d x e) mod lcm(p-1, q-1) = 1*
6. The **private key** is <n, d>

pk

sk

And so e is **not** a divisor of lcm(p-1, q-1)
[Thanks to Josh Perrett for pointing out the omission]

RSA (outline)

7. To **encrypt** a message *m*
   $c = m^e \bmod n$

pk <n, e>

sk <n, d>

8. To **decrypt** a ciphertext *c*
   $m = c^d \bmod n$

Security relies on the difficulty of factoring *n* into *p* and *q* (for large values)

RSA (toy example)

1. Choose any two distinct prime numbers, p & q
2. Compute $n = pq$
3. Select a prime number e between 1 and $lcm(p\text{-}1,q\text{-}1)$
4. The **public key** is <n, e>
5. Compute the number d such that
   $(d \times e) \bmod lcm(p-1, q-1) = 1$
6. The **private key** is <n, d>

Let's choose $p=11,\ q=13$

RSA (toy example)

1. Choose any two distinct prime numbers, *p* & *q*

   p=11, q=13

2. Compute *n* = *pq*

   n = 11 x 13 = 143

3. Select a prime number *e* between 1 and *lcm*(*p*-1,*q*-1)

4. The **public key** is <n, e>

5. Compute the number *d* such that
   (d x e) mod lcm(p-1, q-1) = 1

6. The **private key** is <n, d>

RSA (toy example)

p=11, q=13, n=143

1. Choose any two distinct prime numbers, *p* & *q*

2. Compute *n = pq*

3. Select a prime number *e* between 1 and *lcm*(*p*-1,*q*-1)

4. The **public key** is <n, e>

5. Compute the number *d* such that
   (*d* x *e*) *mod lcm(p-1, q-1) = 1*

6. The **private key** is <n, d>

Technically needs to be **coprime** with the *lcm,* so can't be a divisor of it, and practically should be large

lcm(11-1, 13-1)
least common multiple of 10 and 12

prime number between 1 and 60

Let's say e=17

Known as **Carmichael's totient function**

RSA (toy example)

1. Choose any two distinct prime numbers, *p* & *q*
2. Compute *n = pq*
3. Select a prime number *e* between 1 and *lcm*(*p*-1,*q*-1)
4. The **public key** is <n, e>
5. Compute the number *d* such that (*d x e) mod lcm(p-1, q-1) = 1*
6. The **private key** is <n, d>

p=11, q=13, n=143
e=17

pk <n=143, e=17>

RSA (toy example)

1. Choose any two distinct prime numbers, *p* & *q*

2. Compute *n = pq*

3. Select a prime number *e* between 1 and *lcm*(*p*-1,*q*-1)

4. The **public key** is <n, e>

5. Compute the number *d* such that
   (*d x e) mod lcm(p-1, q-1) = 1*

6. The **private key** is <n, d>

p=11, q=13, n=143
e=17

Where the magic happens...

lcm(11-1, 13-1) = 60
(17 x d) mod 60 = 1
d = 53

17 x 53 = 901
901 mod 60 = 1

The **modular multiplicative inverse** of *e*

RSA (toy example)

1. Choose any two distinct prime numbers, *p* & *q*

2. Compute $n = pq$

3. Select a prime number *e* between 1 and $lcm(p\text{-}1, q\text{-}1)$

4. The **public key** is <n, e>

5. Compute the number *d* such that $(d \times e) \bmod lcm(p-1, q-1) = 1$

6. The **private key** is <n, d>

p=11, q=13, n=143
e=17, d=53

sk <n=143, d=53>

RSA (toy example)

7. To **encrypt** a message *m*
   $c = m^e \bmod n$

   Say our message is m=10

   (all messages can also be represented as numbers)

   $c = 10^{17} \bmod 143$
   $c = 43$

pk <n=143, e=17>

RSA (toy example)

8. To **decrypt** a ciphertext *c*

   $m = c^{d} \bmod n$

   Our ciphertext is c=43

   m = 43^{53} mod 143

   m = 10

sk <n=143, d=53>

RSA (toy example)

In practice:

- The values used in this example are trivial to factor, primes in real use are much, much larger.

- (Remember, decomposing *n* into prime factors *p* and *q* would allow you to easily calculate *d*)

- Efficient computation complicates the picture given here. (*d* is actually broken down into several quantities)

PGP

PGP

Public-key encryption was invented and patented by Rivest, Shamir and Adleman. They called the first algorithm RSA (1977).

In 1991, Phil Zimmerman put a free implementation on the internet, called Pretty Good Privacy (PGP).

This led to a dispute with RSA and the US Government - who gave up in 1996.

GPG

Once PGP was definitely legal, it went commercial.

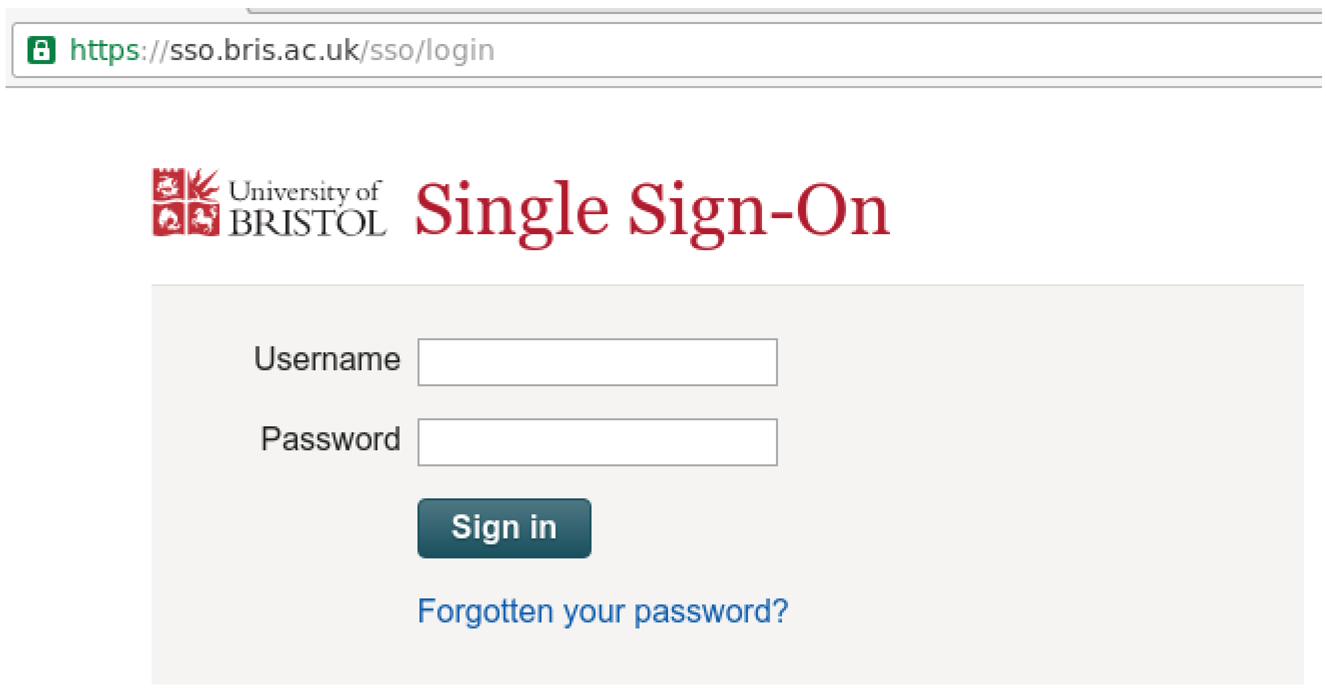
GPG / GnuPG (Gnu Privacy Guard) took over as the go-to free encryption software and is widely used, especially in the Linux world.

GPG is a command-line (console) program; several graphical (with menus and buttons) frontends exist for it.

key exchange

Problem

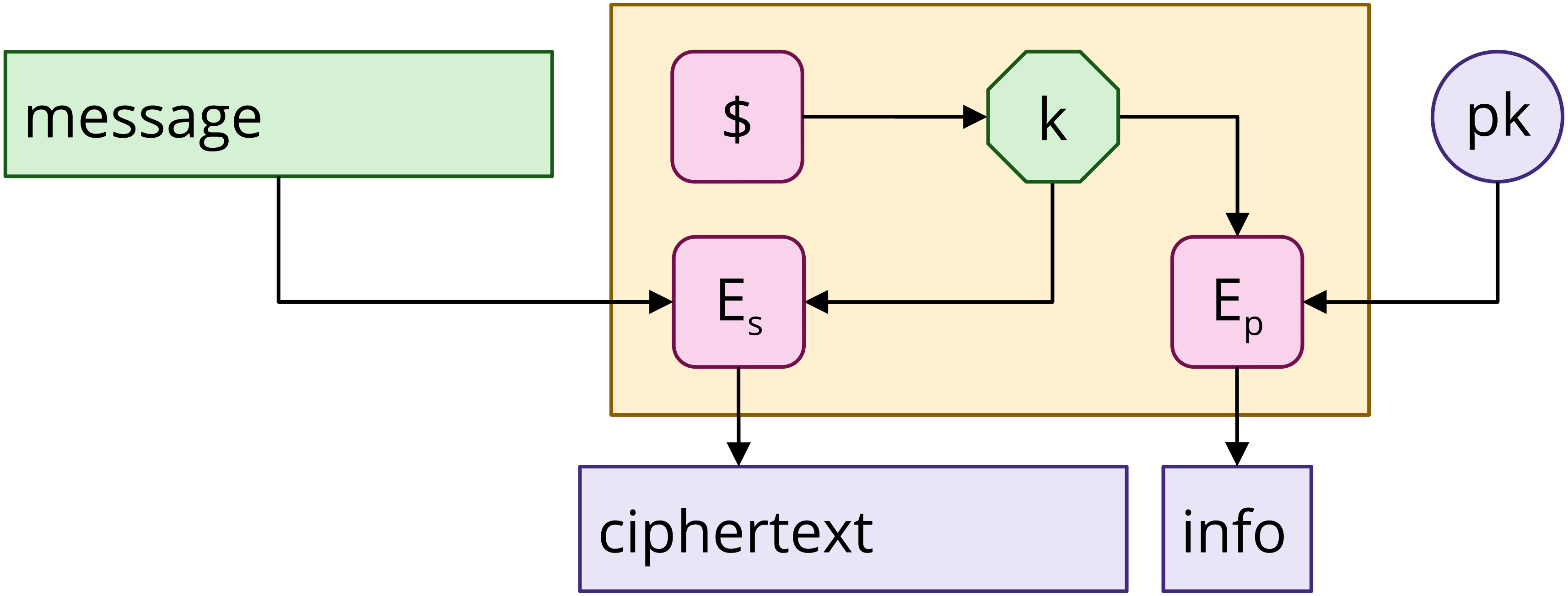
You’ve never been to this site before … where did you get the key from?



Hybrid encryption

Public key encryption is much slower than symmetric encryption.

How to use it efficiently:



Key exchange

Diffie, Hellman 1976: can two people agree on a shared secret key over the telephone, without anyone listening in learning it?

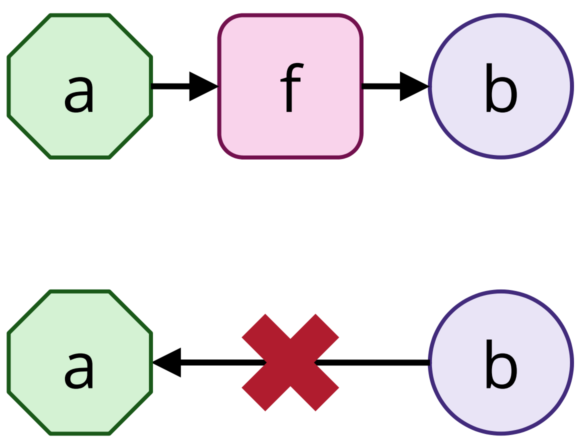
The answer is “obviously no” - but yes.

Magic functions

Imagine we had a function that's both

one-way (easy to compute, impossible to invert)

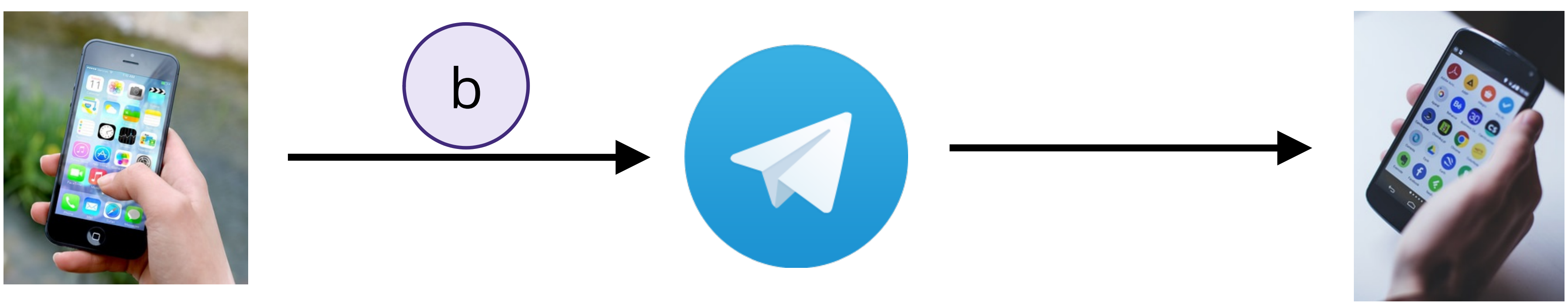
linear: **f(a+b) = f(a) + f(b), f(c · a) = c · f(a)**



Diffie-Hellman key exchange

secret: a

b = f(a)



Diffie-Hellman key exchange

secret: a

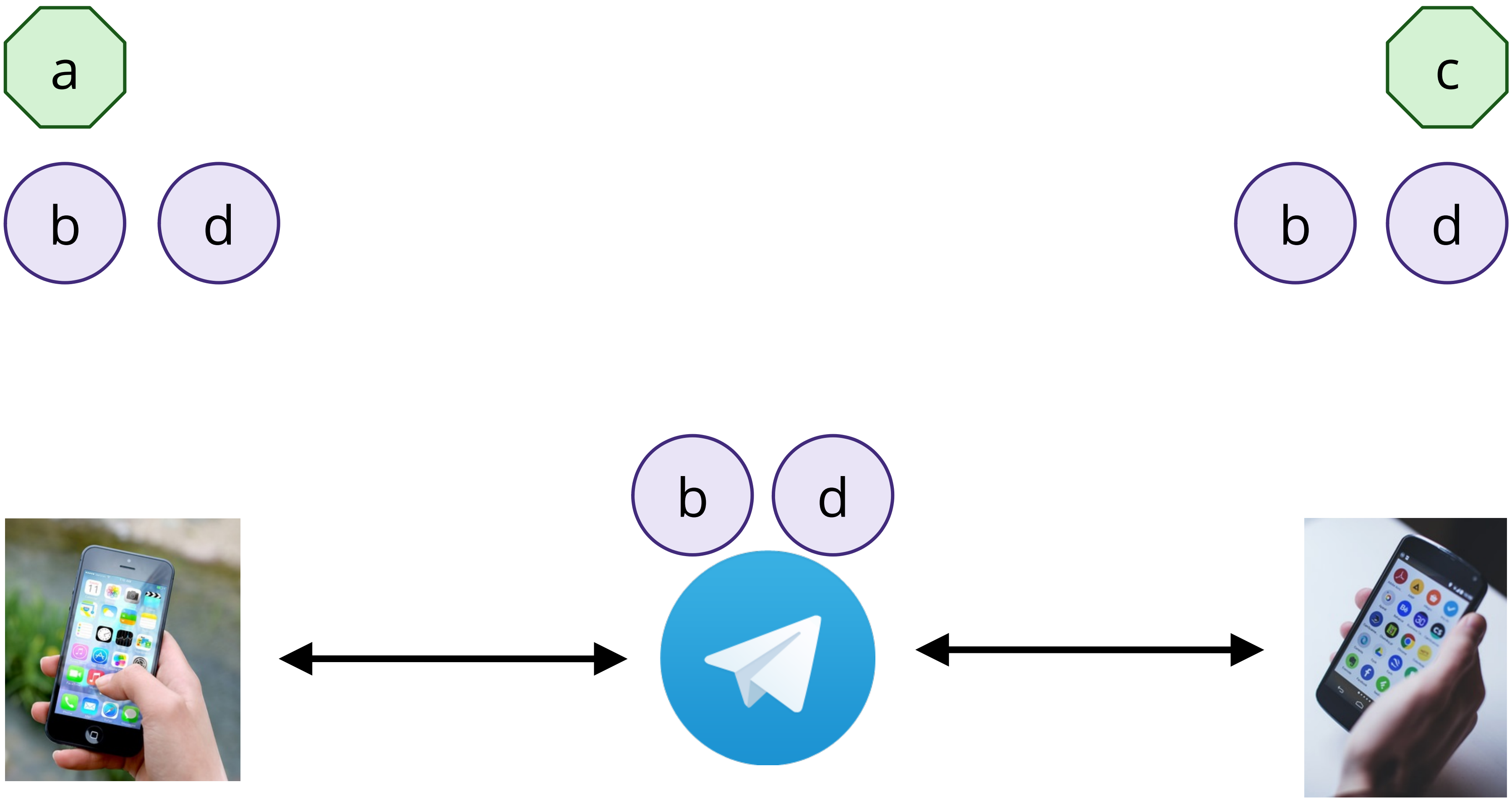
$b = f(a)$

secret: c

$d = f(c)$

d

The idea



Key derivation

a

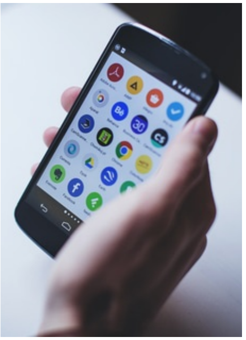
b d

$k = a \cdot d$

c

b d

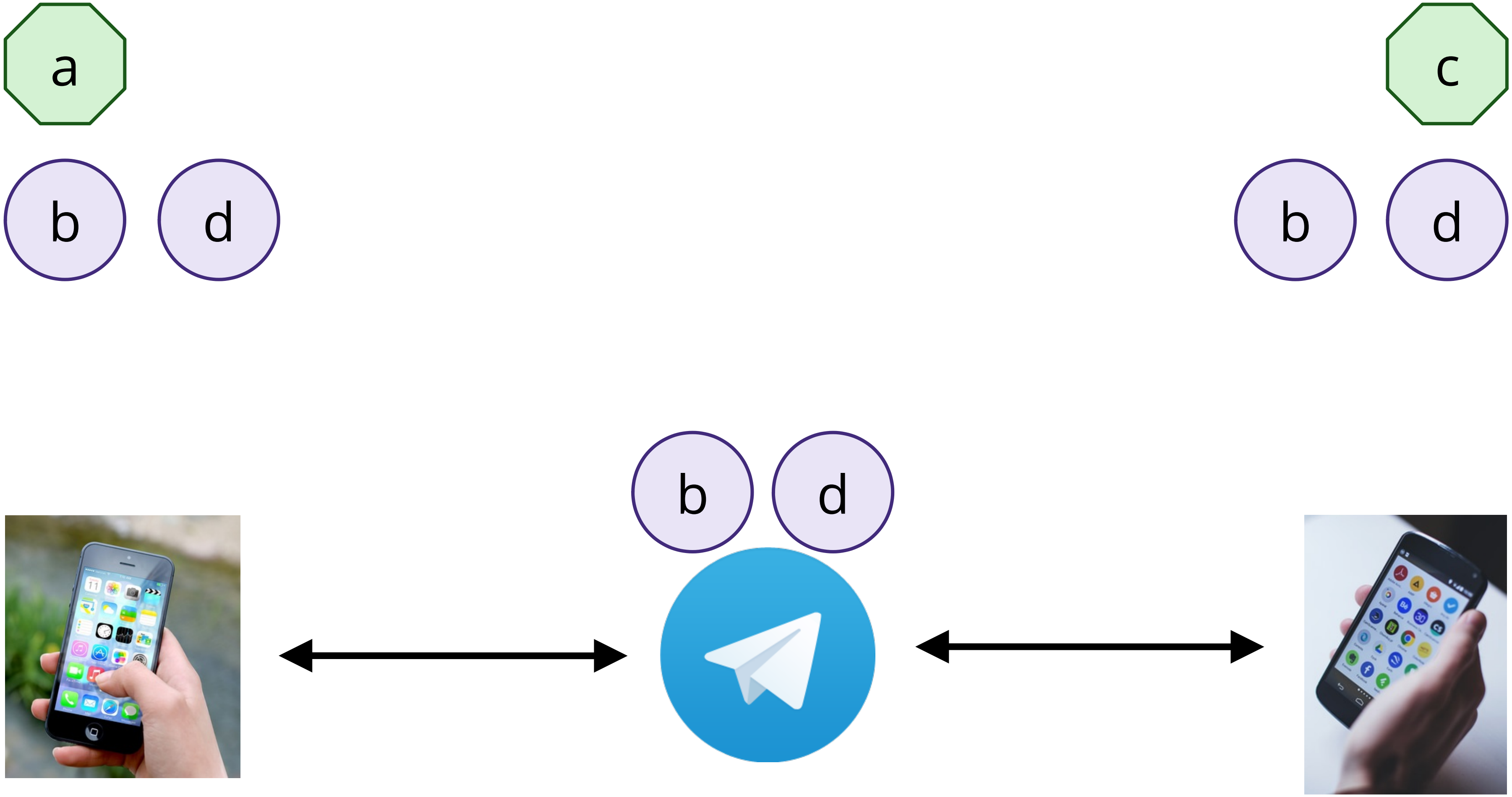
$k = c \cdot b$



Correctness

a · d = a · f(c) = f(a · c) = c · f(a) = c · b

Security



Key derivation

actually …

$k = H(a \cdot d)$ $\quad$ $k = H(c \cdot b)$

- Helps with security.
- Necessary to get a random-looking bitstring **k**.
- Gives you lots of keys for one exchange:
  $\mathbf{k_i = H(i, a \cdot d)}$.

◊ Does this exist?

Do suitable vector spaces with linear one-way functions exist?

candidate one: $V = Z^*_p$, $f(x) = g^x \pmod p$.

(g is a fixed, public base value)

candidate two: V = elliptic curve, $f(x) = x \cdot P$.

(P is a fixed, public point on the curve)

More detail in future cryptography courses!

Key lengths

security (bits)	RSA keylength	EC keylength
128	3248	256
192	7936	384
256	15424	512

applies for Z*p too

elliptic curves give better sizes

Google example

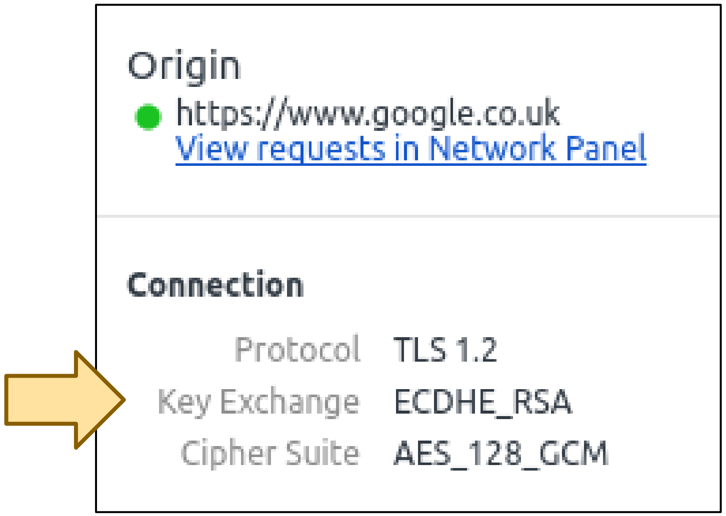
Google uses ECDHE_RSA.

ECDH = Elliptic Curve Diffie-Hellman.

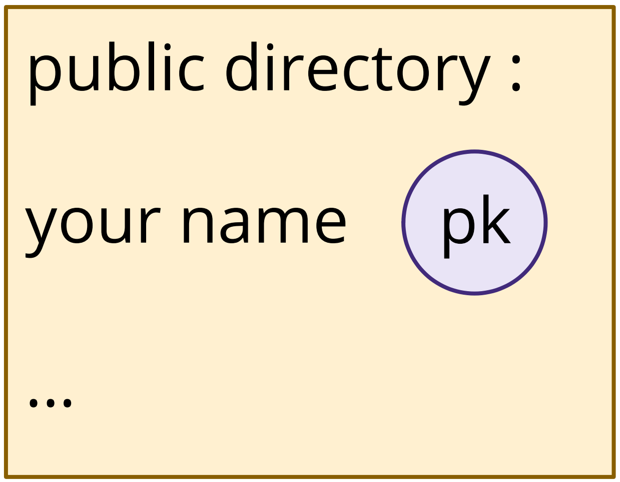
Key Exchange using EC

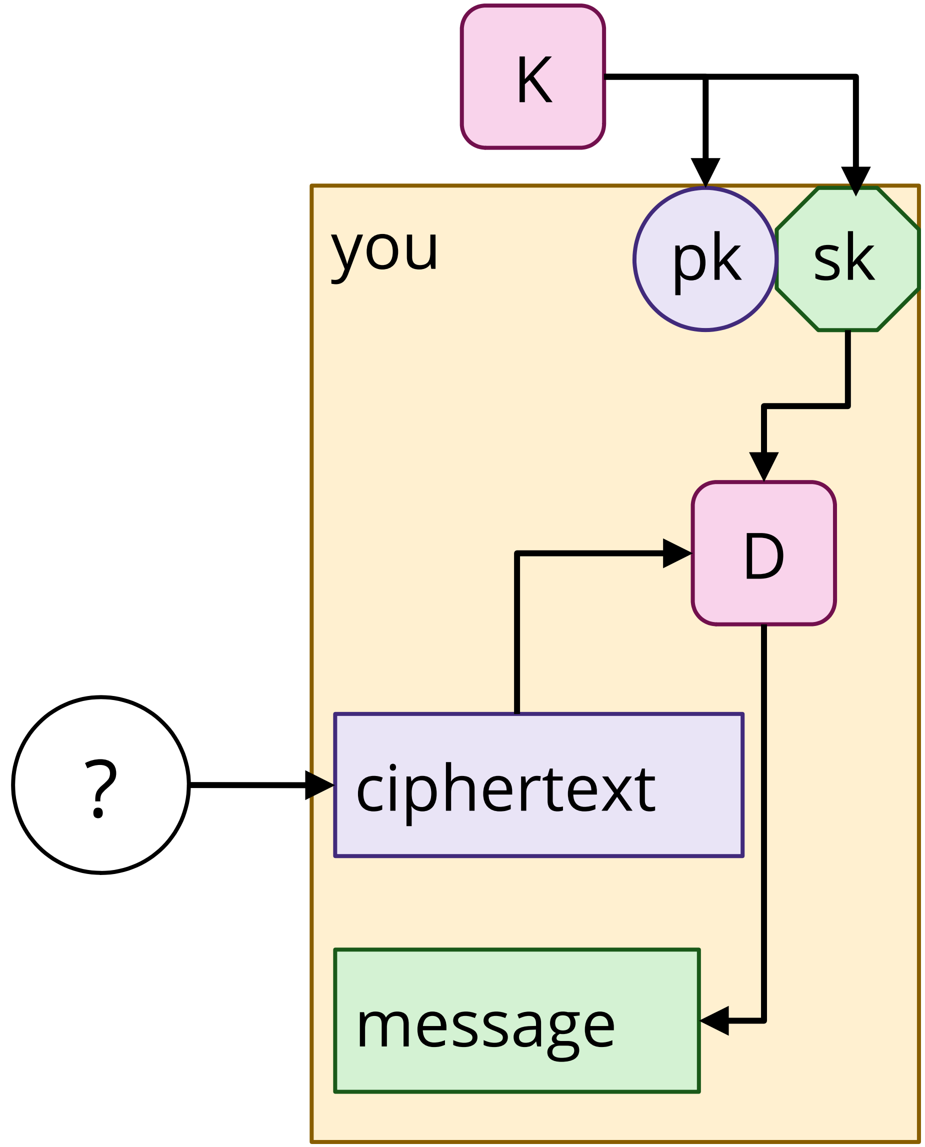
Messages encrypted with AES (symmetric block cipher)

So what is RSA doing here?



Public-key encryption





Public-key encryption on its own offers absolutely no authenticity - anyone with your PK (which is supposed to be public!) can send you encrypted messages.