Before the break:

When using Internet technologies, we are confronted with two fundamental questions:

- How to hide *what* is communicated? ◀
- How to hide *who* communicates?

...in the face of an opponent that has *total knowledge of all the IP traffic involved*.

Live example: manually decrypting a ciphertext

"HG"

Live example: manually decrypting a ciphertext

What has just happened?

- The receiver applied a decryption algorithm...

  "On receiving a message, perform the following two steps:

  ① Replace each character by its alphabetical successor: an 'A' becomes a 'B', a 'B' a 'C', etc.

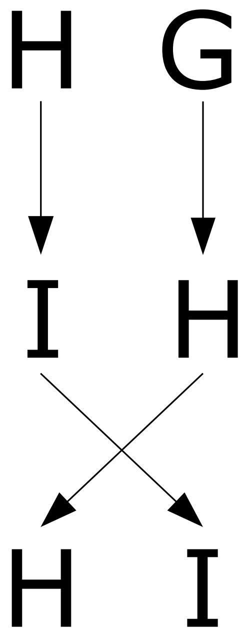
  ② Reverse the order: put the last character first, followed by the one-before-last, etc."

- ...to decrypt a ciphertext:

  "HG" ①→ "IH" ②→ "HI"

Live example: manually decrypting a ciphertext

This was an example of a substitution-permutation cipher:



Substitution: replace the instances of one symbol by those of another.

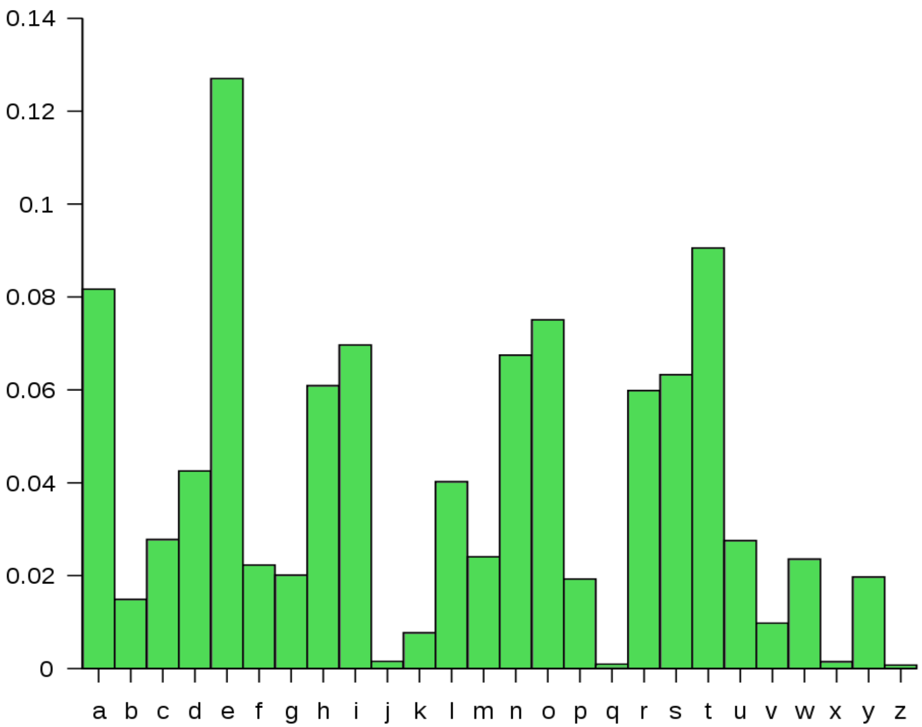
Permutation: change the order in which symbol instances are placed.

Live example: manually decrypting a ciphertext

- The example encryption method uses permutation & substitution to produce an overall ciphertext that is different from the original, plaintext message.

- What remains constant however, is that *symbols of the same type* still end up in the ciphertext as *symbols of the same type*.

- This means our ciphertexts will be vulnerable to letter frequency analysis →



↑ Letter frequencies in English.
Original by Nandhp at Wikimedia Commons.

- To make this attack harder to do, you can group symbol instances together into blocks: a block cipher.

Live example: automatic encryption and decryption (using JavaScript)

...However, we will now first *automate* encryption:

- Writing code in JavaScript: the "+*x substitution cipher*".

Live example: automatic encryption and decryption (using JavaScript)

- From the example code, we could further automate the crack ( ) function by adding a dictionary lookup test...

Live example: automatic encryption and decryption (using JavaScript)

- *Everyday intuition:*
  "Cracking an encrypted message should be hard."

- *Now, as code:*

```
ciphertext = encrypt (plaintext, key); // should finish quickly
plaintext = decrypt (ciphertext, key); // should finish quickly
plaintext = crack (ciphertext); /* should finish only after the
                                   end of the universe */
```

⇒ *So, more precisely:*
The cracking algorithm should consist of many more computational steps than the encryption and decryption algorithms.

Live example: automatic encryption and decryption (using JavaScript)

- In the example, our repeated decryption attempts were done using the following code:

```
possible_plaintext = encrypt (ciphertext, -1 * key);
```

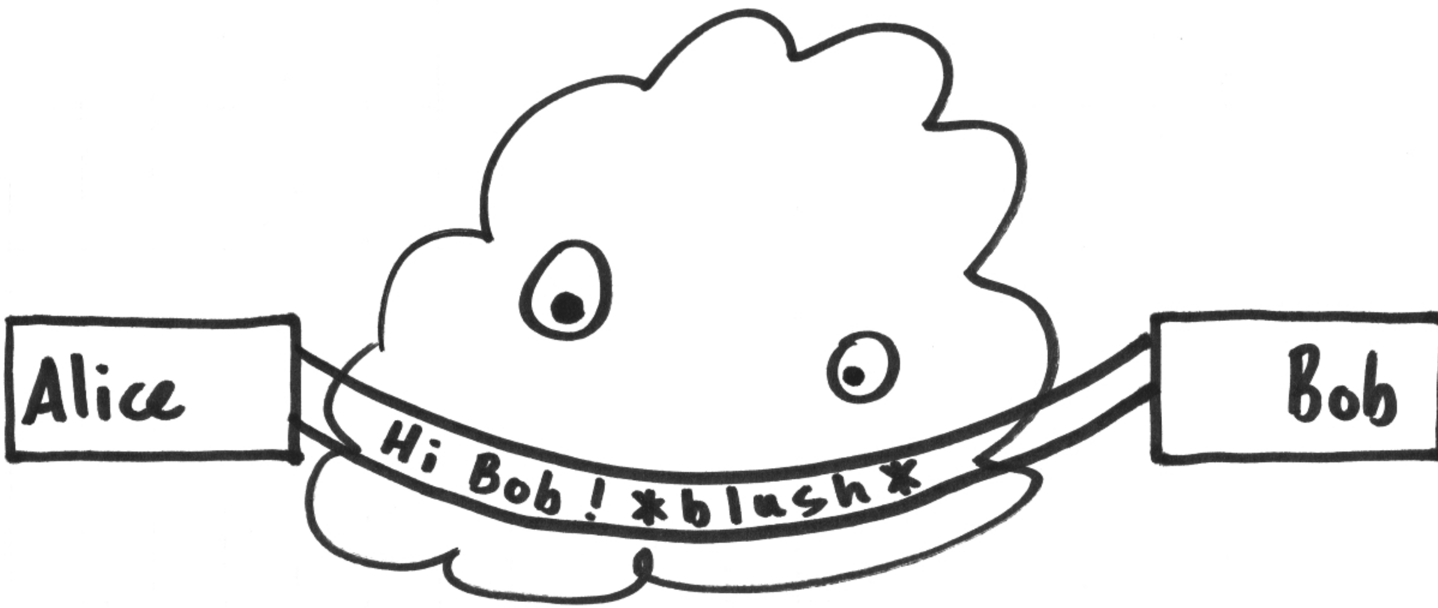
- This is an example of symmetric-key encryption:

  The same key that is used to encrypt the plaintext
  is also used to decrypt the resulting ciphertext.

- Asymmetric-key encryption is also used, e.g. in public key cryptography:

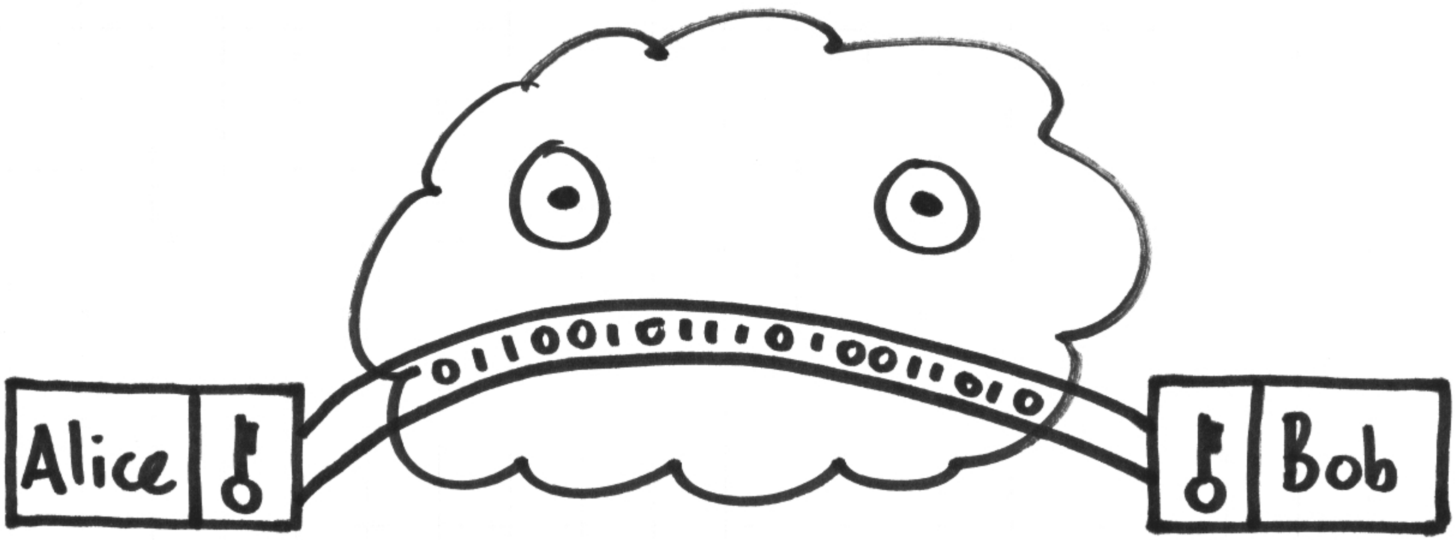
  Here, *anybody can encrypt* using a publicly accessible key;
  but *only the receiver can decrypt* using a second, private key.

Symmetric-key encryption



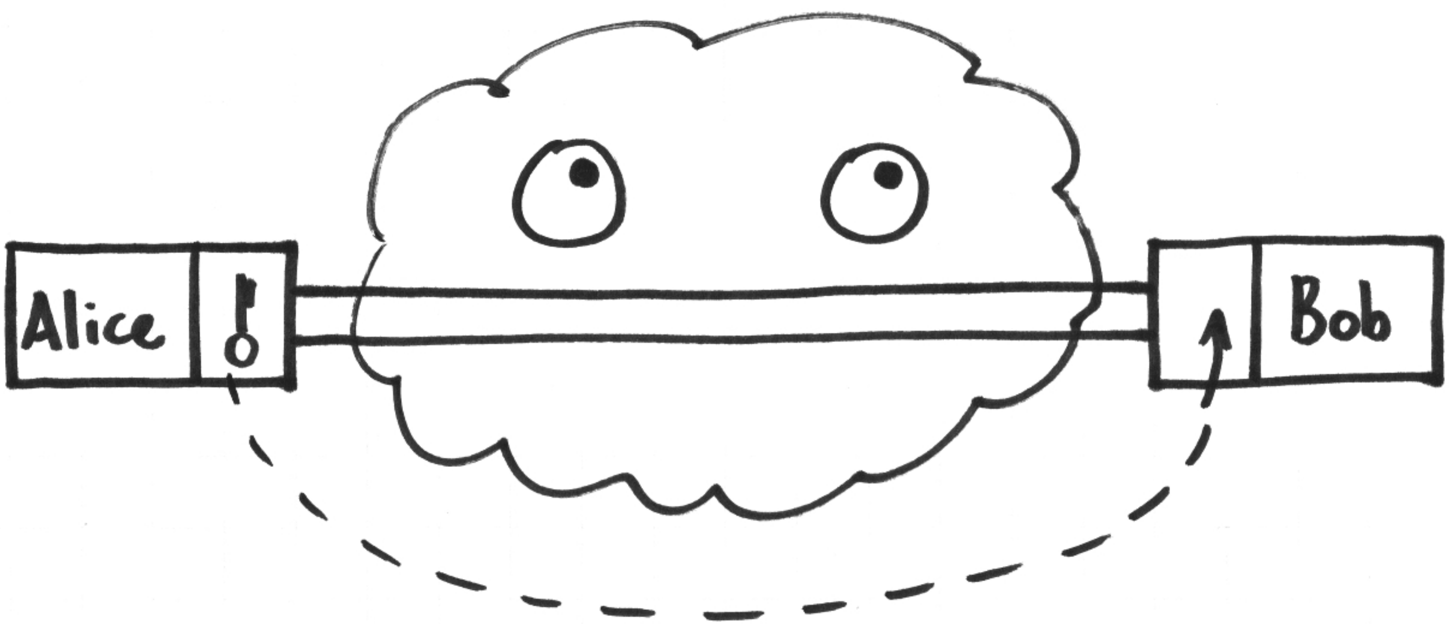
Can be used...

Symmetric-key encryption



...to hide *what* is communicated over the Internet.

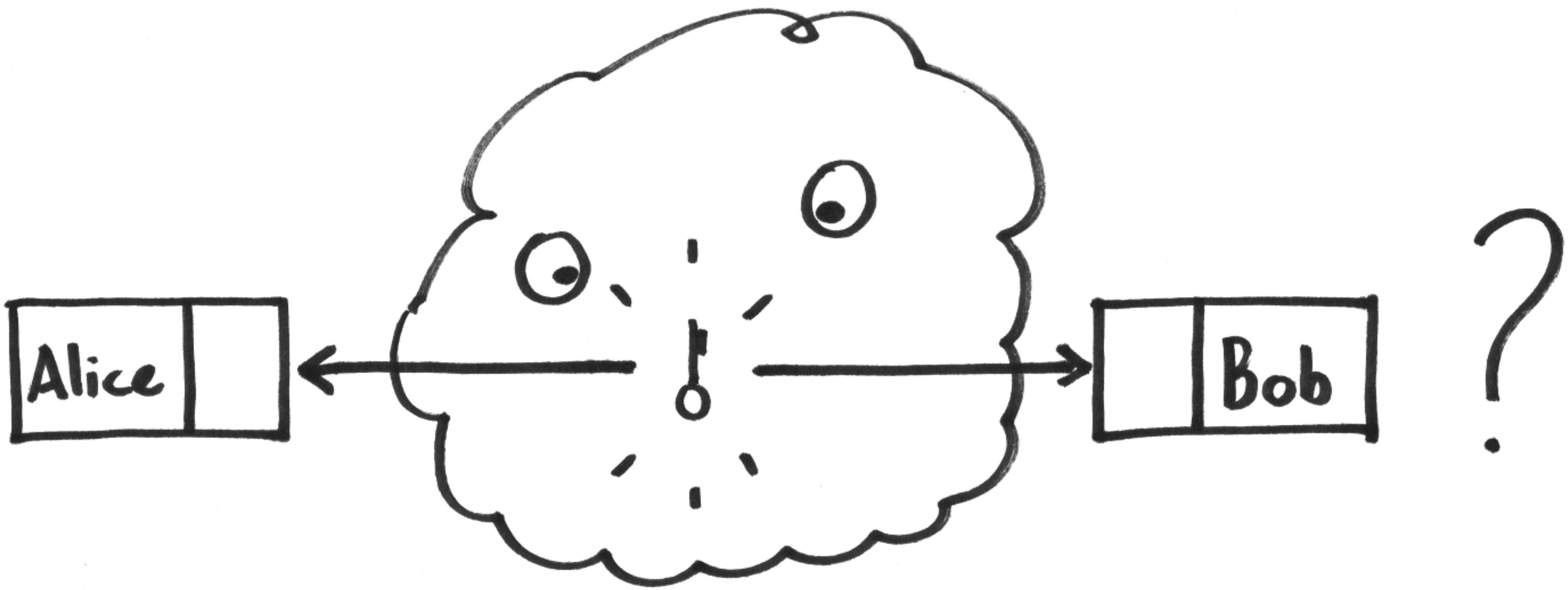
Symmetric-key encryption



- All we need is a covert, secure channel, to exchange the key.

↑ ...Wait a minute!

Symmetric-key encryption



- Remember: our opponent has total knowledge of all IP traffic.

  ⇒ *Chicken-and-egg problem:* We need a shared key to communicate securely – but we first need to communicate securely, to obtain a shared key. → ...What to do?

Tools to make a secret key with everyone looking

- What to do? We need to grasp three concepts:
  - “An encryption key is a number.”
  - One-way functions.
  - Quasi-commutative functions.

Tools to make a secret key with everyone looking

- A numerical function f ( ) may combine two input numbers a and b to form one new output number o:

  o = f (a, b); // think of JavaScript here

- A commutative function gives the same output when a and b are swapped.
  - E.g. doing an addition, (a + b) == (b + a) is always true.
  - The general case: “f (a, b) == f (b, a) is always true.”

- A quasi-commutative function gives the same output in the following argument swap:
  - f (f (a, b), c) == f (f (a, c), b)
  - E.g. doing two additions: (a + b) + c == (a + c) + b.
  - So: it does not matter if you first combine a with b, then c; or first with c, then b.

Tools to make a secret key with everyone looking

- A numerical function f ( ) may combine two input numbers a and b to form one new output number o:

  o = f (a, b); // think of JavaScript here

- For a one-way function:
  - When you know a, b, and f ( ), computing o takes *only a few steps*.
  - But even when you know o, a, and f ( ), computing b takes *very many steps*.
  - So: computation is quick only in one direction.

    ⇒ f ( ) is “hard to crack” based on its output.

Diffie-Hellman key exchange

- Everyone knows the function f ( ) and the constant number g.

- Alice and Bob each secretly pick a random number: a and b.

- Alice privately computes f (g, a), then sends the resulting value to Bob.

- Bob privately computes f (g, b), then sends the resulting value to Alice.

- Everybody can see and record these values!

- But since f ( ) is a one-way function, a and b still remain secret.

Diffie-Hellman key exchange

- Bob then uses the value of f (g, a) to compute f (f (g, a), b).

  The output is known just to Bob, because only he knows b.

- Alice uses the value of f (g, b) and computes f (f (g, b), a).

  The output is known just to Alice: only she knows a.

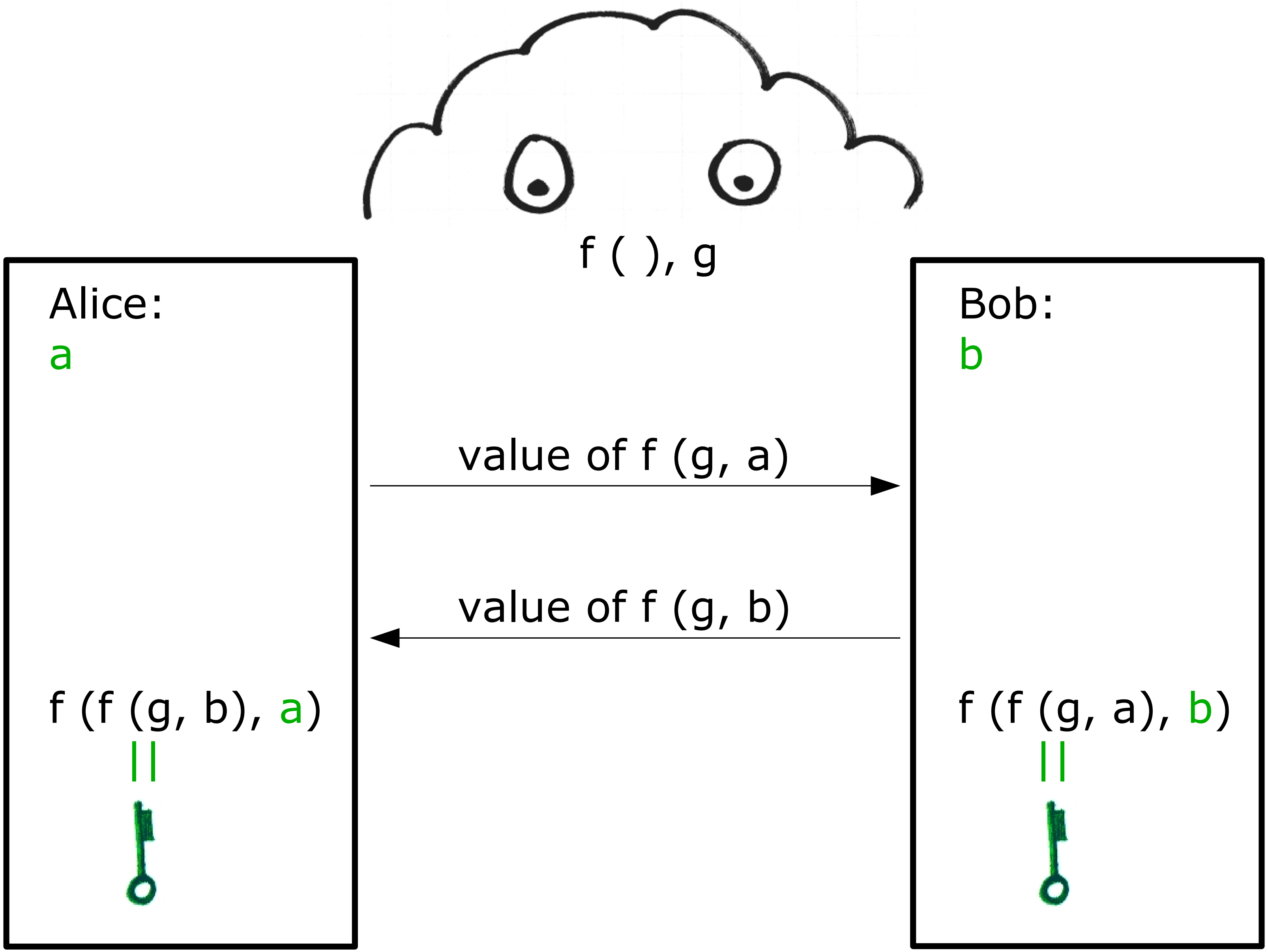
- But wait – *Bob and Alice actually computed the same value!*

- As f ( ) is quasi-commutative: f (f (g, a), b) == f (f (g, b), a).

- Bob and Alice now *both* know this number *no one else knows*.

  ⇒ They can use it as a symmetric key.

Diffie-Hellman key exchange, recapitulated



TLS: Transport Layer Security

- *Application layer* protocol which extends transport layer protocols with encryption.

- Diffie-Hellman here is an optional key exchange mechanism.

- TLS has been implemented on top of TCP (but also UDP).

- HTTP Secure (HTTPS): HTTP via TLS (instead of TCP).