CMSC/Math 456: Cryptography (Fall 2022)

Lecture 9

Daniel Gottesman

Administrative

Reminder: Problem Set #3 is due Thursday (Sep. 29) at noon.

Some notes about the problem set:

- Remember to name your file "attack.py"
- The IV and key are lists of independent random bytes of an appropriate length. In particular, it is possible for values to repeat.
- There were some bugs in the autograder which have been fixed. The autograder has been rerun and some scores changed.
- Hint: In order to solve problem 1 for all lengths, your attack function will need to look at the IV provided to it, not just the list x.

Solution set #2 is available on ELMS.

This class is being recorded

Two Questions

Question: How is a cryptographer like a magician?

Two Questions

Question: How is a cryptographer like a magician?

Answer: A cryptographer never reveals their secrets.

Question: How is a cryptographer *not* like a magician?

Two Questions

Question: How is a cryptographer like a magician?

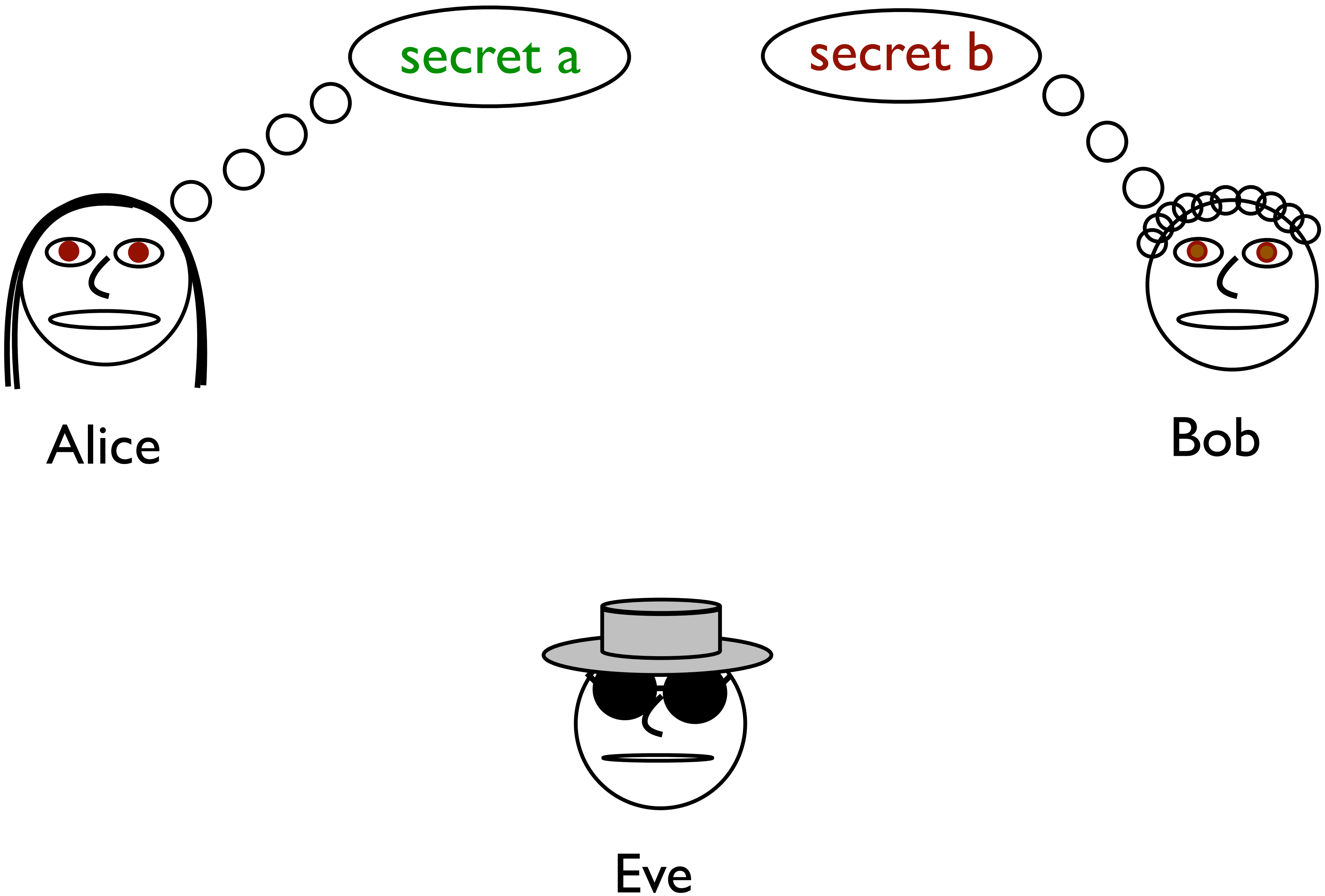
Answer: A cryptographer never reveals their secrets.

Question: How is a cryptographer *not* like a magician?

Answer: A cryptographer will tell you how they did it.

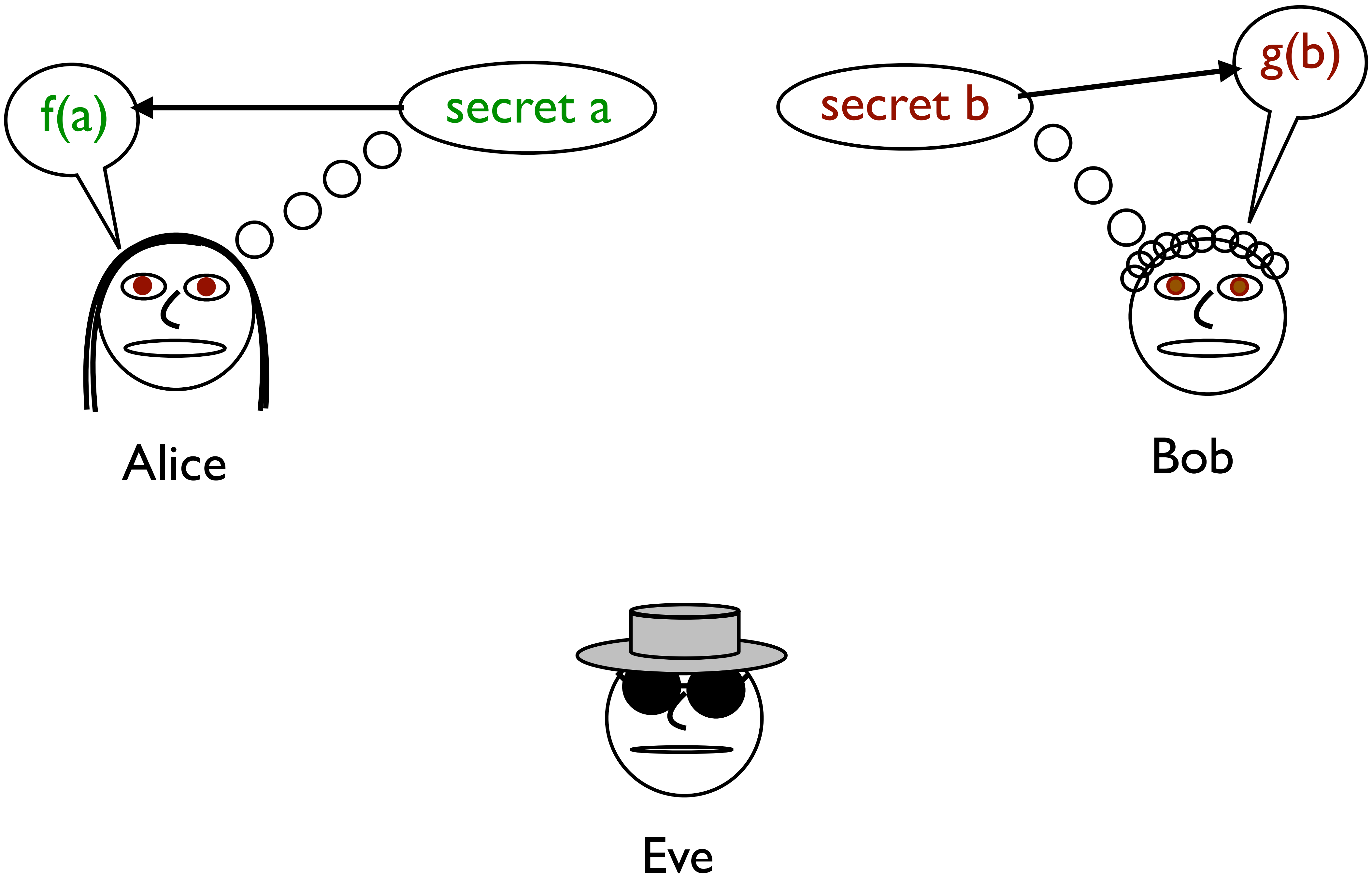
Let us now perform a cryptographic magic trick.

Key Exchange



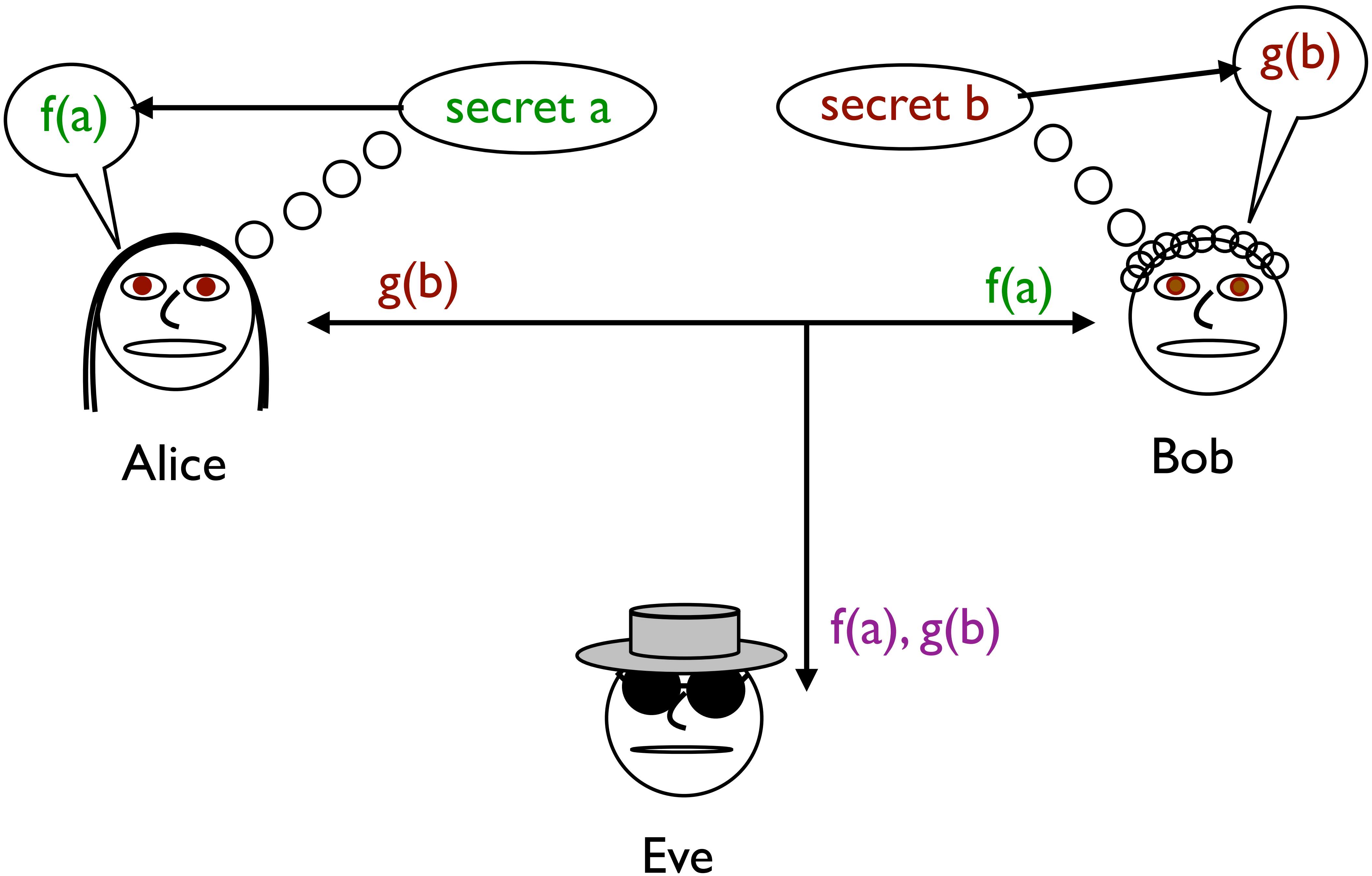
This class is being recorded

Key Exchange

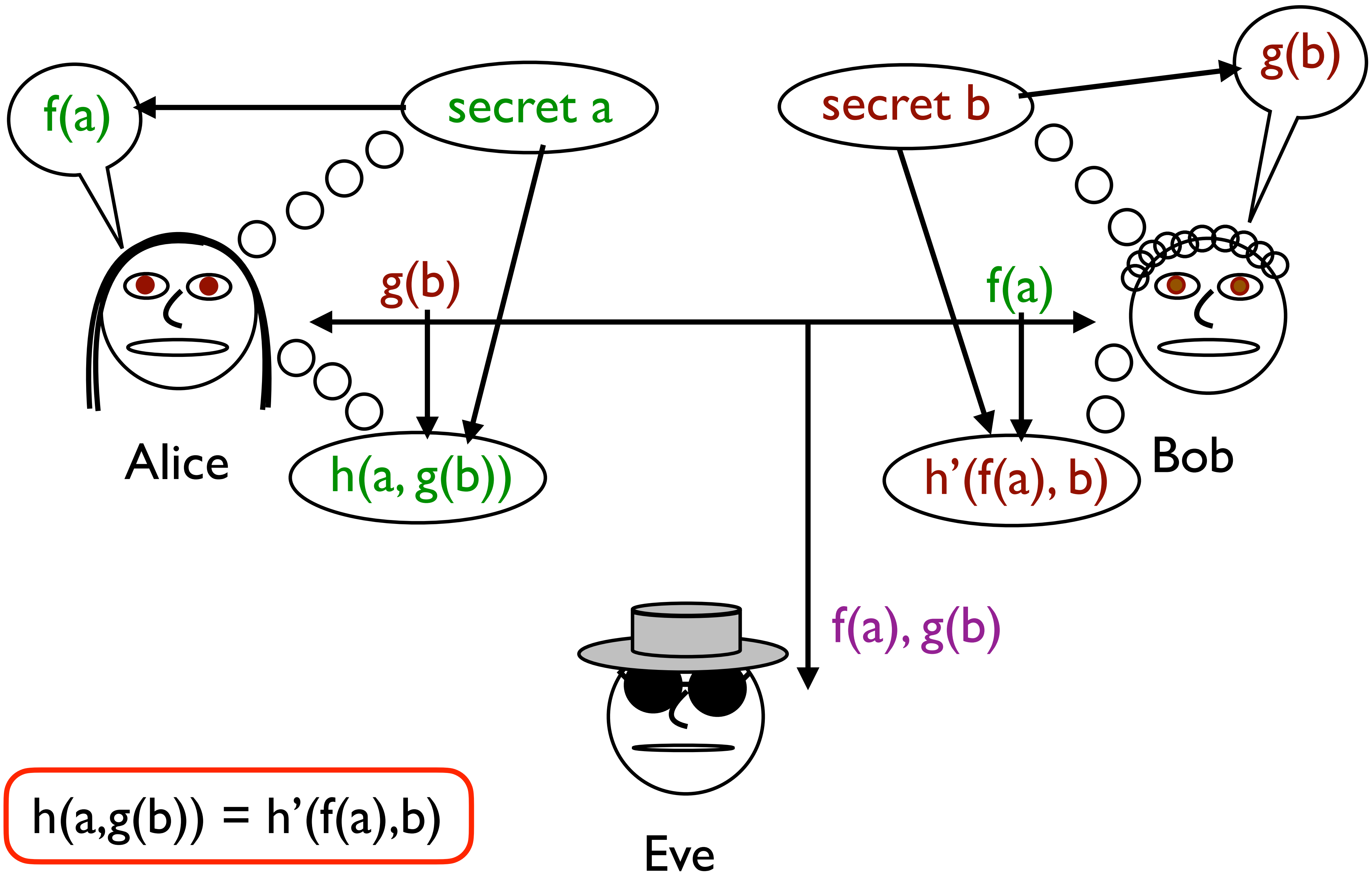


This class is being recorded

Key Exchange



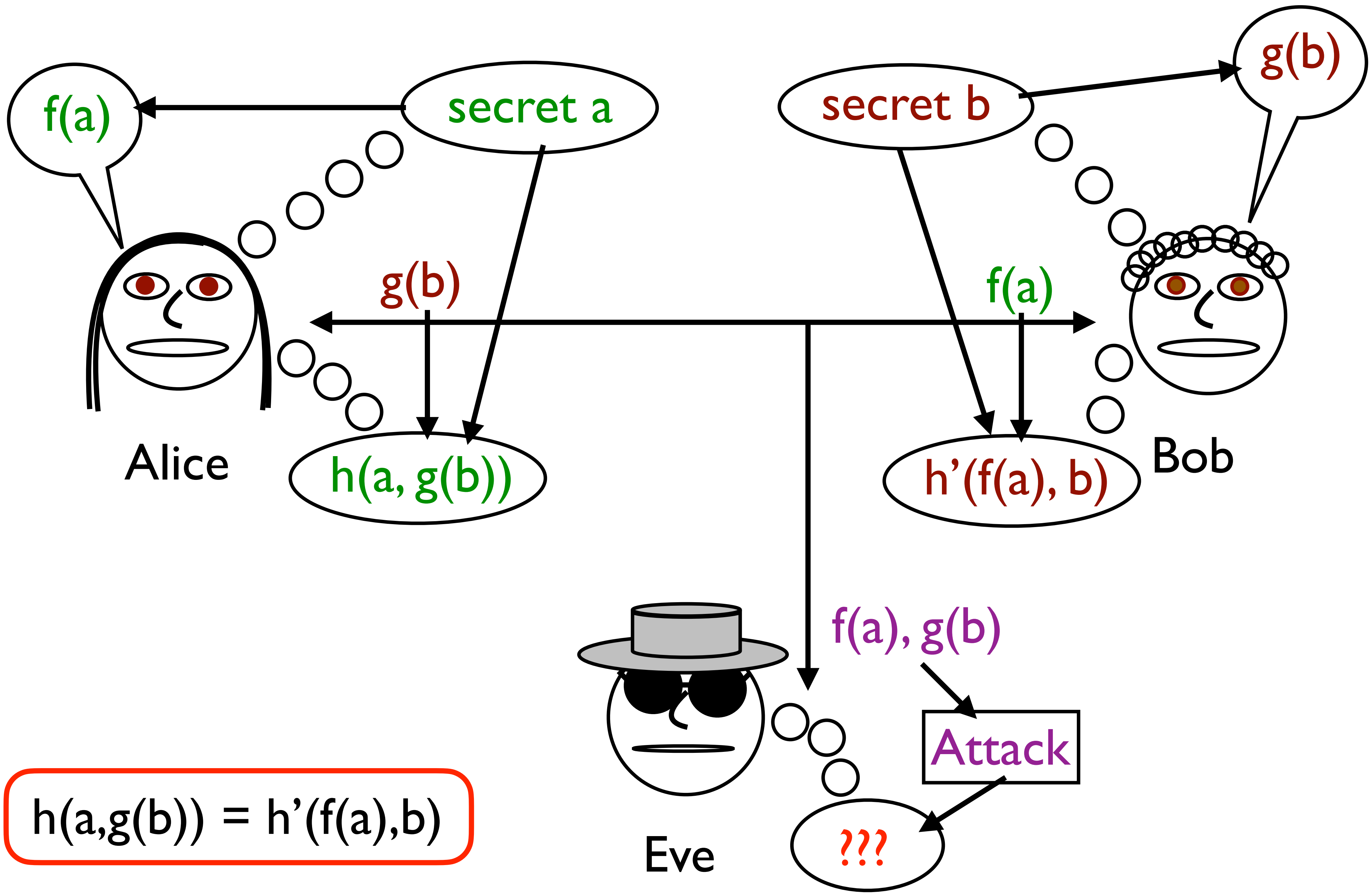
Key Exchange



$$h(a,g(b)) = h'(f(a),b)$$

This class is being recorded

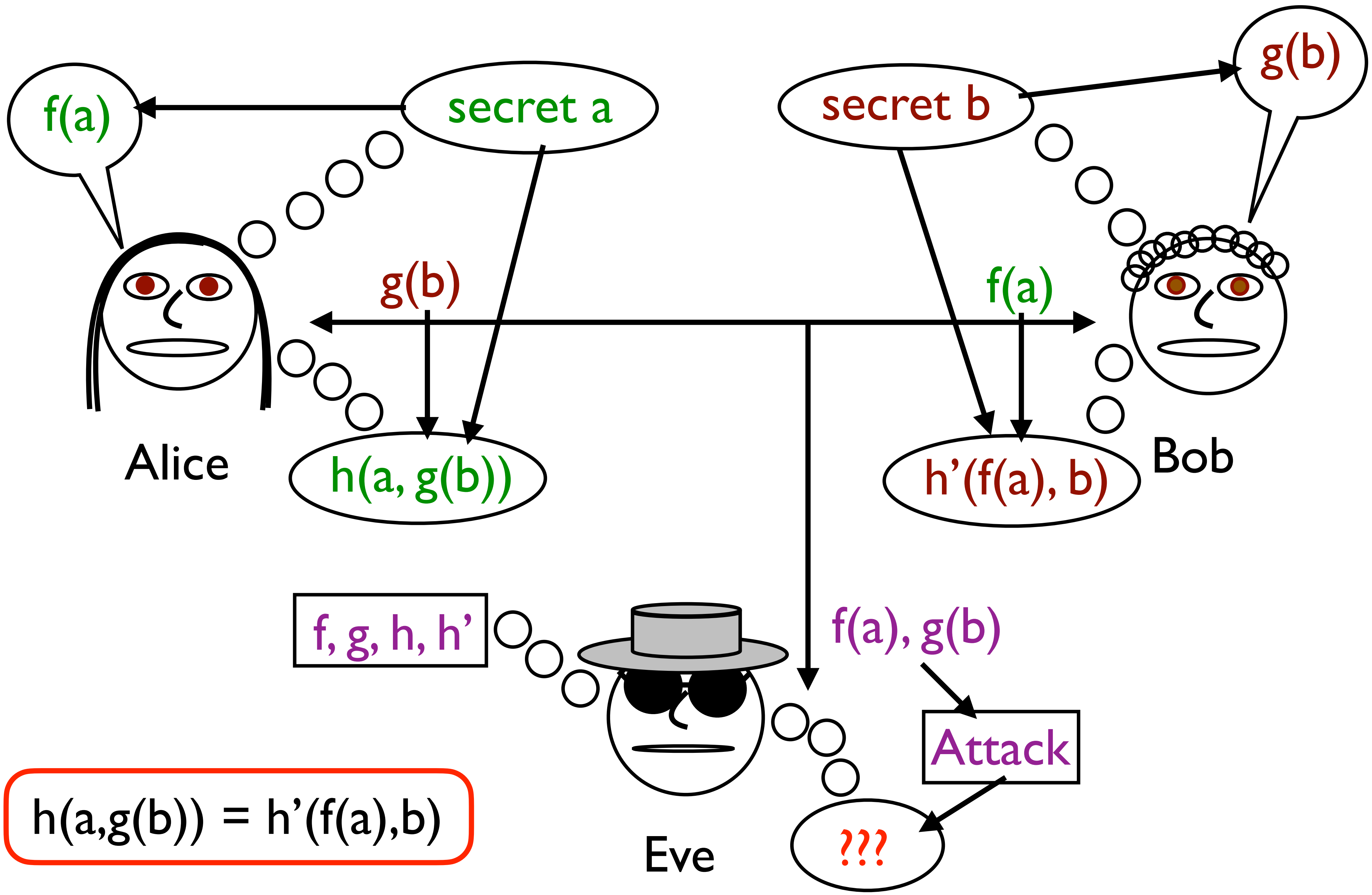
Key Exchange



$$h(a,g(b)) = h'(f(a),b)$$

This class is being recorded

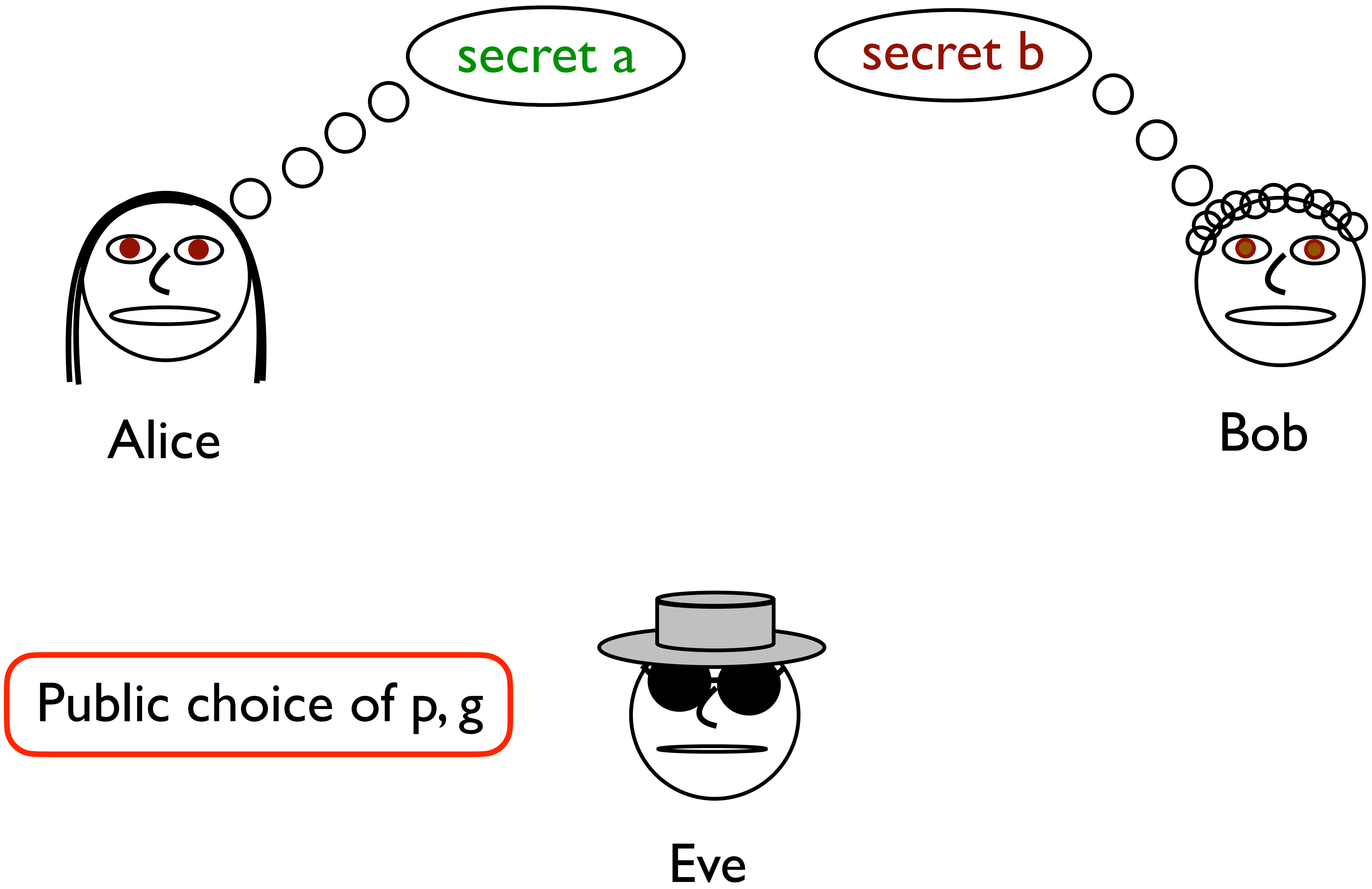
Key Exchange



h(a,g(b)) = h'(f(a),b)

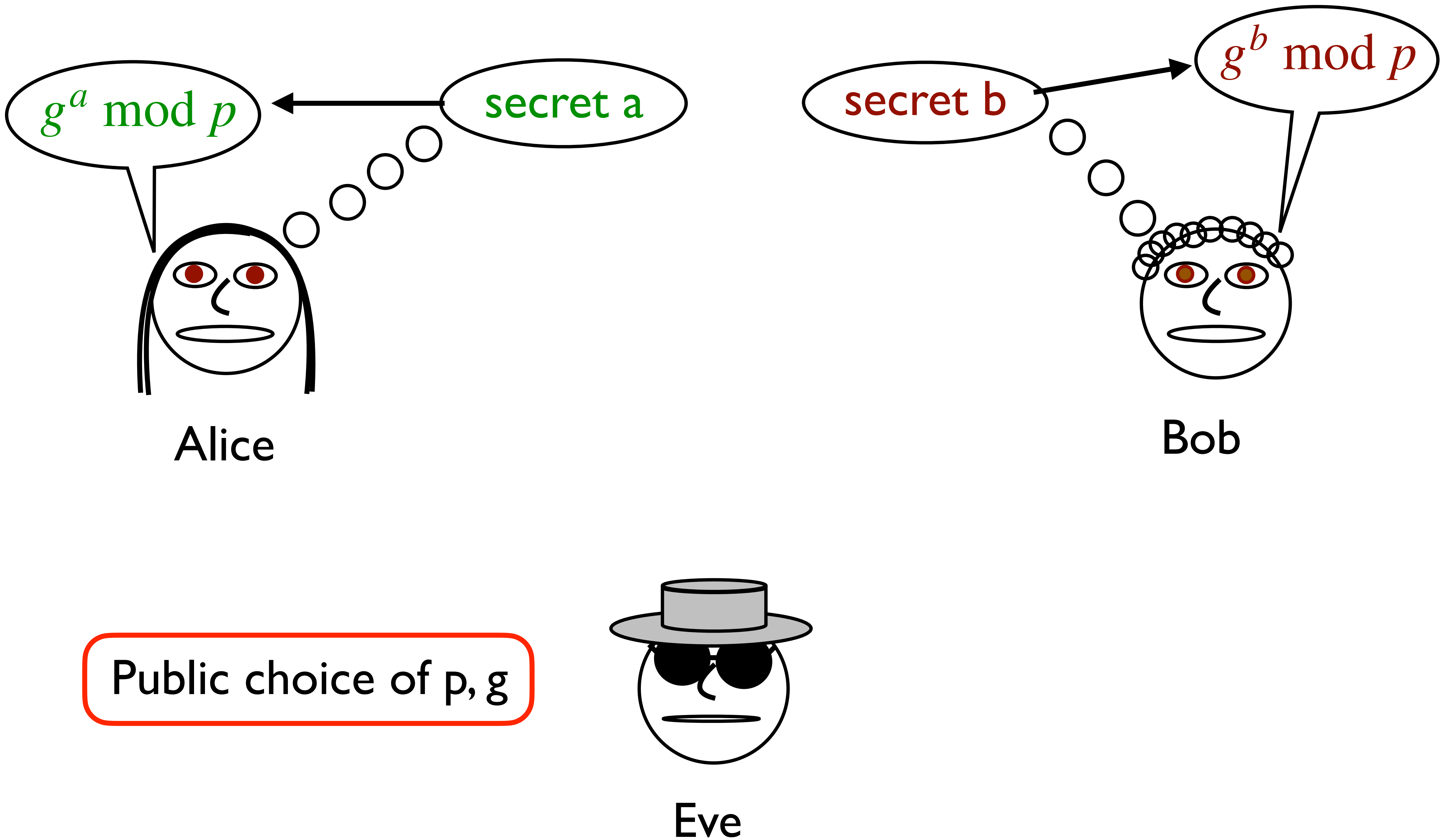
This class is being recorded

Diffie-Hellman Key Exchange



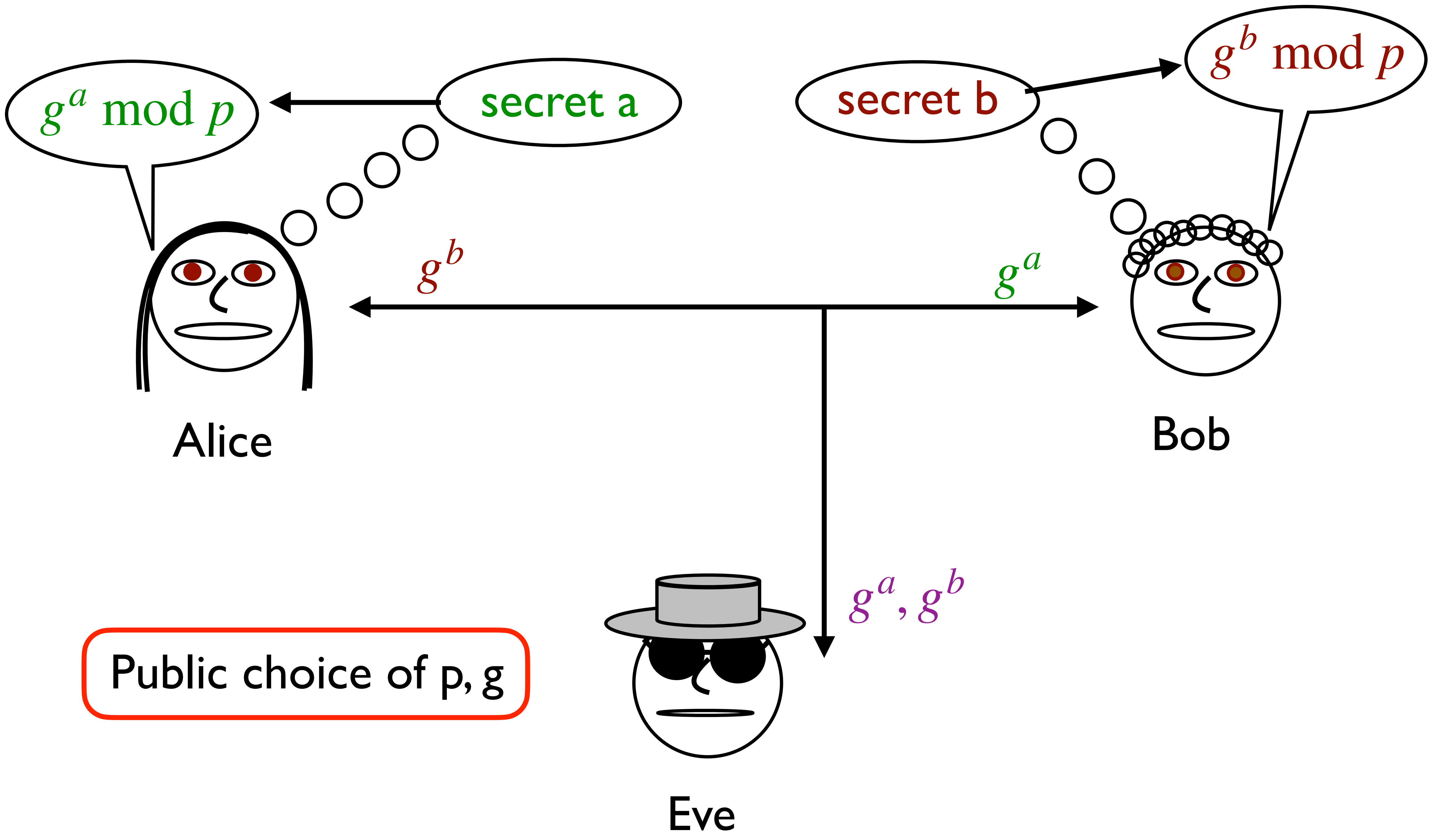
This class is being recorded

Diffie-Hellman Key Exchange

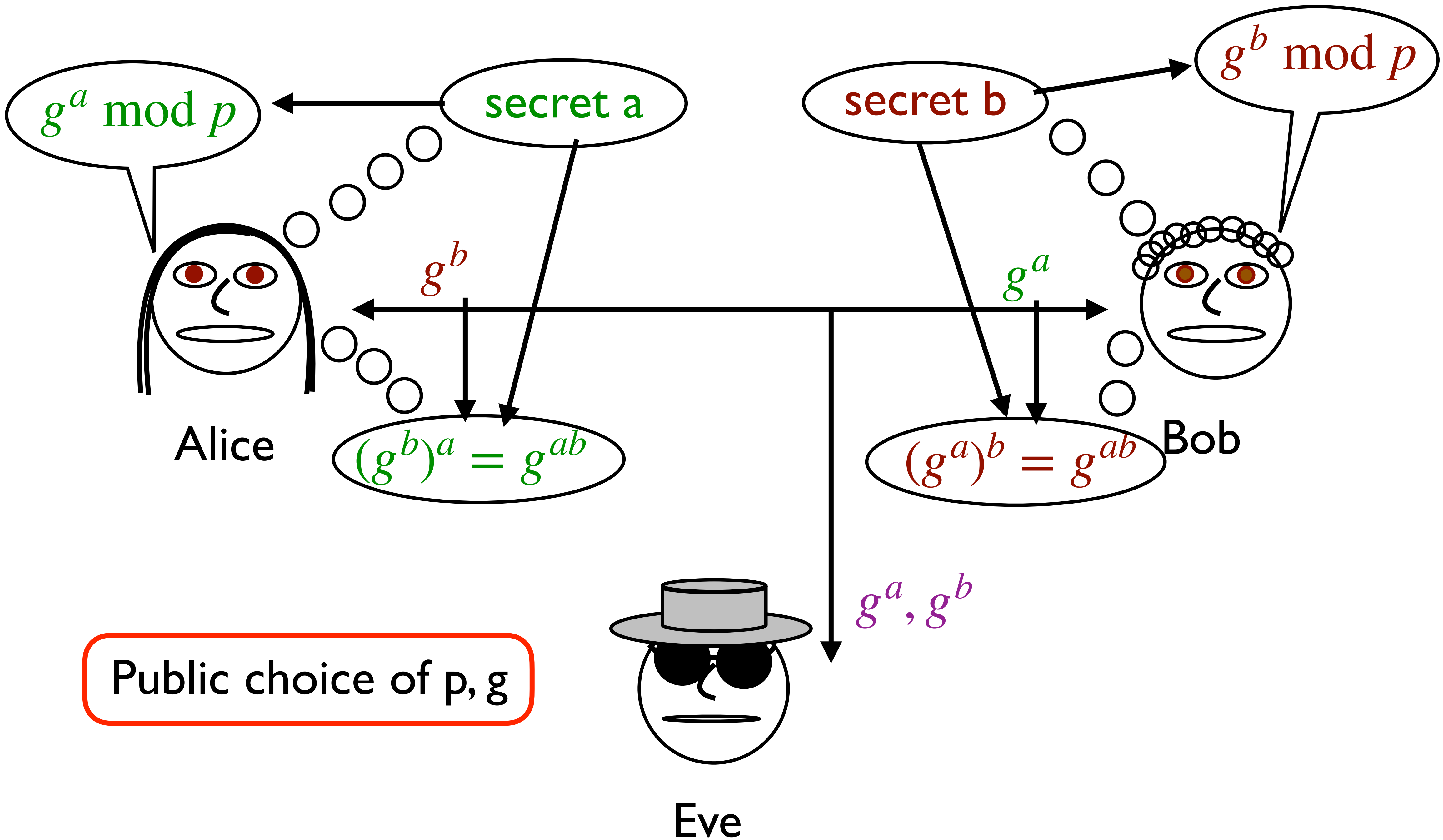


This class is being recorded

Diffie-Hellman Key Exchange



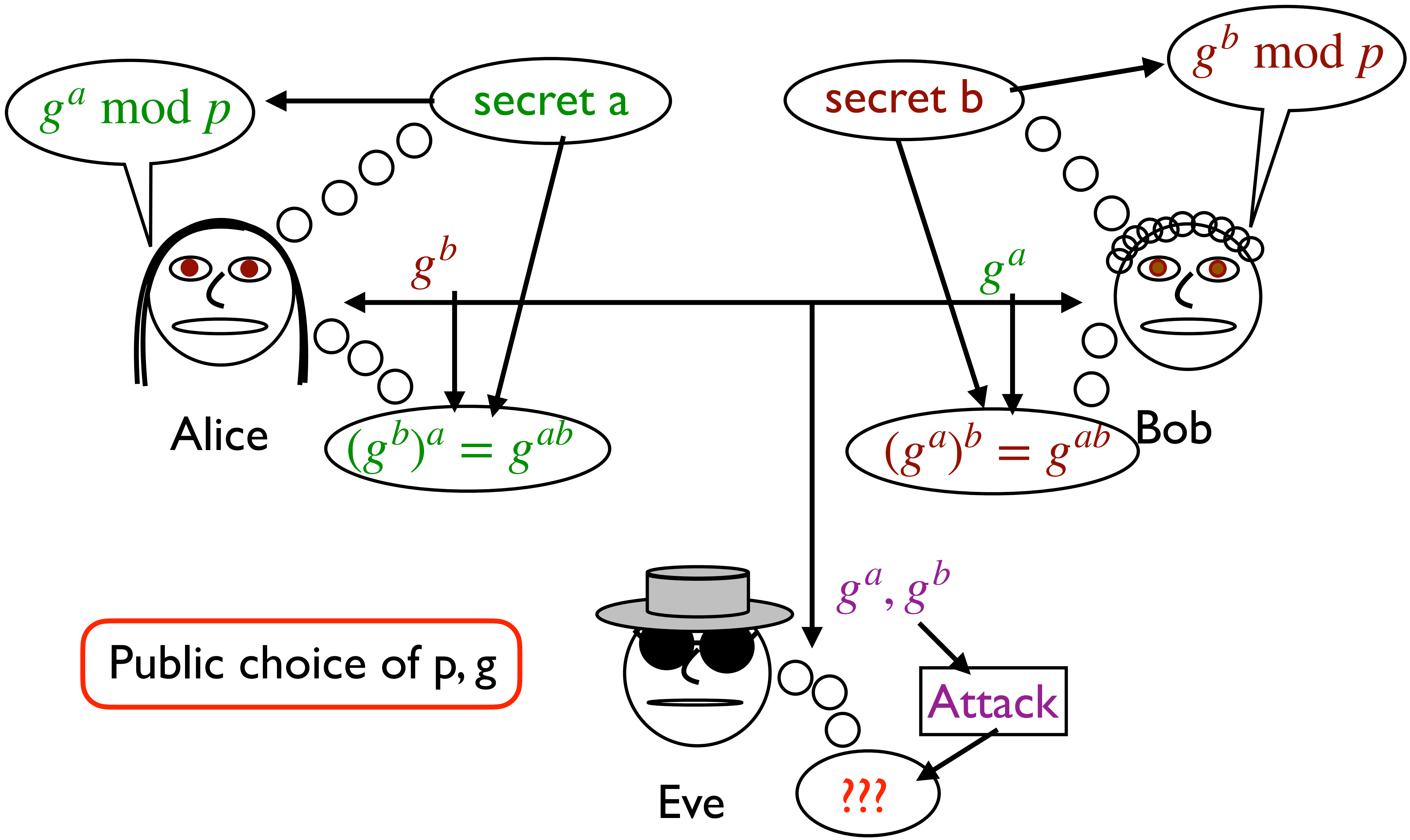
Diffie-Hellman Key Exchange



Public choice of p, g

This class is being recorded

Diffie-Hellman Key Exchange



Public choice of p, g

This class is being recorded

Diffie-Hellman Magic Demonstration

We will use $p = 71$ and $g = 65$.

Diffie-Hellman Magic Demonstration

We will use p = 71 and g = 65.

Alice: Choose & record a secret number a from 0 to 70.
Compute

$$A = 65^a \bmod 71$$

Bob: Choose &record a secret number b from 0 to 70.
Compute

$$B = 65^b \bmod 71$$

Diffie-Hellman Magic Demonstration

We will use p = 71 and g = 65.

Alice: Choose & record a secret number a from 0 to 70.
Compute

$$A = 65^a \bmod 71$$

Bob: Choose &record a secret number b from 0 to 70.
Compute

$$B = 65^b \bmod 71$$

Alice and Bob: announce A and B to the class.

Diffie-Hellman Magic Demonstration

We will use p = 71 and g = 65.

Alice: Choose & record a secret number a from 0 to 70.
Compute

$$A = 65^a \bmod 71$$

Bob: Choose &record a secret number b from 0 to 70.
Compute

$$B = 65^b \bmod 71$$

Alice and Bob: announce A and B to the class.

Alice: Compute $B^a \bmod 71$ and write it down secretly.

Bob: Compute $A^b \bmod 71$ and write it down secretly.

Do not reveal them until I say to.

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition:

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition: Works the same. E.g.:

$$(36 + 58) + 15 \bmod 71 = 38 = 36 + (58 + 15) \bmod 71$$

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition: Works the same. E.g.:

$$(36 + 58) + 15 \bmod 71 = 38 = 36 + (58 + 15) \bmod 71$$

Subtraction:

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition: Works the same. E.g.:

$$(36 + 58) + 15 \bmod 71 = 38 = 36 + (58 + 15) \bmod 71$$

Subtraction: Works the same. E.g.:

$$36 - 58 = 49 \bmod 71$$

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition: Works the same. E.g.:

$$(36 + 58) + 15 \bmod 71 = 38 = 36 + (58 + 15) \bmod 71$$

Subtraction: Works the same. E.g.:

$$36 - 58 = 49 \bmod 71$$

Multiplication:

This class is being recorded

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition: Works the same. E.g.:

$$(36 + 58) + 15 \bmod 71 = 38 = 36 + (58 + 15) \bmod 71$$

Subtraction: Works the same. E.g.:

$$36 - 58 = 49 \bmod 71$$

Multiplication: Works the same. E.g.:

$$36 * 58 \bmod 71 = 29 = 58 * 36 \bmod 71$$

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition: Works the same. E.g.:

$$(36 + 58) + 15 \bmod 71 = 38 = 36 + (58 + 15) \bmod 71$$

Subtraction: Works the same. E.g.:

$$36 - 58 = 49 \bmod 71$$

Multiplication: Works the same. E.g.:

$$36 * 58 \bmod 71 = 29 = 58 * 36 \bmod 71$$

Division:

Modular Arithmetic

To understand what is going on with Diffie-Hellman and how one might attack it or make it harder to attack, we need to know a lot more about modular arithmetic.

Addition: Works the same. E.g.:

$$(36 + 58) + 15 \bmod 71 = 38 = 36 + (58 + 15) \bmod 71$$

Subtraction: Works the same. E.g.:

$$36 - 58 = 49 \bmod 71$$

Multiplication: Works the same. E.g.:

$$36 * 58 \bmod 71 = 29 = 58 * 36 \bmod 71$$

Division: There are some issues. E.g.:

$$35/58 \bmod 60 = ?$$

This question has no answer. $\nexists x \text{ s.t. } 58 * x = 35 \bmod 60$

Modular Divison

But $36/58 \bmod 71$ *is* well-defined:

$$58 * 60 = 1 \bmod 71$$

Thus,

$$36/58 \bmod 71 = 36 * 60 \bmod 71 = 30$$

How do we determine if division is allowed or not?

$$a/b = c \bmod N \iff a = bc + kN$$

Suppose there is some p such that $p \mid b$ and $p \mid N$. Then the right-hand side of the equation on the right is also a multiple of p. Thus, division by b is only possible if a is a multiple of p as well.

What about if b and N are *relatively prime*? (I.e., they have no common factors.)

Finding GCDs

Definition: Let gcd(a,b) be *greatest common divisor* of positive integers a and b: namely, the largest integer c such that $c \mid a$ and $c \mid b$. Note that if a and b are relatively prime, gcd(a,b) = 1.

Theorem: For any two positive integers a and b, there exists a polynomial-time algorithm to find X and Y such that

$$aX + bY = \gcd(a, b)$$

Note: If $d < \gcd(a, b)$, then $aX + bY \neq d$ for any integers X,Y.

Proof: The proof is an analysis of the algorithm to find X and Y. This is Euclid's algorithm.

> Euclid's algorithm appeared in Euclid's *Elements* in around 300 BCE. That makes it one of the world's oldest algorithms!

This class is being recorded

Euclid's Algorithm Concept

Suppose we want to find c = gcd(a,b).

We know $c \mid a$ and $c \mid b$. Can we find another smaller number that is also a multiple of c?

If $a > b$, then a' = a-b is smaller than a and must still be a multiple of c.

If we keep subtracting one number from the other, our pair of numbers will get steadily smaller until eventually we get down to c.

Example:

a = 58
b = 36
c = 2 (but we don't know that yet)

a-b = 22
(still a multiple of c)

36 - 22 = 14

22 - 14 = 8

14 - 8 = 6

8 - 6 = 2

6 is a multiple of 2, so we are done.

Euclid's Algorithm Refinements

When we subtract off b from a, the result might still be bigger than b. Instead we should take $a \bmod b$, which means subtract off as many b's as we can. This will give us a number a' which is less than b, so next time we reduce b instead.

Euclid's Algorithm Refinements

When we subtract off b from a, the result might still be bigger than b. Instead we should take $a \bmod b$, which means subtract off as many b's as we can. This will give us a number a' which is less than b, so next time we reduce b instead.

To get the coefficients X and Y, we should also keep track of how *many* b's we subtracted:

$$a' = a - Y_0 b$$

Euclid's Algorithm Refinements

When we subtract off b from a, the result might still be bigger than b. Instead we should take $a \bmod b$, which means subtract off as many b's as we can. This will give us a number a' which is less than b, so next time we reduce b instead.

To get the coefficients X and Y, we should also keep track of how *many* b's we subtracted:

$$a' = a - Y_0 b$$

At each step, this will allow us to write our current replacements for a and b in the form $aX_i + bY_i$.

Euclid's Algorithm Refinements

When we subtract off b from a, the result might still be bigger than b. Instead we should take $a \bmod b$, which means subtract off as many b's as we can. This will give us a number a' which is less than b, so next time we reduce b instead.

To get the coefficients X and Y, we should also keep track of how *many* b's we subtracted:

$$a' = a - Y_0 b$$

At each step, this will allow us to write our current replacements for a and b in the form $aX_i + bY_i$.

In particular, if our current pair is $a_i = aX_i + bY_i$ and $b_i = aX_i' + bY_i'$, and we subtract m_i copies of b_i, then

$$a_{i+1} = a_i - m_i b_i = a(X_i - m_i X_i') + b(Y_i - m_i Y_i')$$

Euclid's Algorithm Refinements

When we subtract off b from a, the result might still be bigger than b. Instead we should take $a \bmod b$, which means subtract off as many b's as we can. This will give us a number a' which is less than b, so next time we reduce b instead.

> To get the coefficients X and Y, we should also keep track of how *many* b's we subtracted:
>
> $$a' = a - Y_0 b$$

At each step, this will allow us to write our current replacements for a and b in the form $aX_i + bY_i$.

> In particular, if our current pair is $a_i = aX_i + bY_i$ and $b_i = aX_i' + bY_i'$, and we subtract m_i copies of b_i, then
>
> $$a_{i+1} = a_i - m_i b_i = a(X_i - m_i X_i') + b(Y_i - m_i Y_i')$$

We don't need to keep a_i and b_i separate: We can combine them into a single sequence r_i.

Euclid's Algorithm

Let $r_0 = a$ and $r_1 = b$. Assume $a > b$.

$i = 1, X_0 = 1, Y_0 = 0, X_1 = 0, Y_1 = 1$

Repeat:

$$r_{i+1} = r_{i-1} \bmod r_i$$

$$m_i = \lfloor r_{i-1}/r_i \rfloor$$

$$X_{i+1} = X_{i-1} - m_i X_i$$

$$Y_{i+1} = Y_{i-1} - m_i Y_i$$

$$i = i + 1$$

Until $r_i = 0$

Output:

$$\gcd(a, b) = r_{i-1}$$

$$X = X_{i-1}, Y = Y_{i-1}$$

This class is being recorded

Example:

$$r_0 = 57, r_1 = 22$$

$r_2 = 13,$
$X_2 = 1, Y_2 = -2$

$r_3 = 9,$
$X_3 = -1, Y_3 = 3$

$r_4 = 4,$
$X_4 = 2, Y_4 = -5$

$r_5 = 1,$
$X_5 = -5, Y_5 = 13$

$$r_6 = 0$$

$\gcd(57,22) = 1,$
$1 = -5 \cdot 57 + 13 \cdot 22$

Euclid's Algorithm Analysis

At every iteration of the algorithm, the following statements are true:

$$0 \le r_i < r_{i-1}$$

$$r_i = aX_i + bY_i$$

$$\gcd(a, b) \mid r_i$$

If these statements are true for i, the statements also hold true for i+1 (by the arguments before). They are true for i=0 and thus we prove by induction that the statements are true for all i.

Euclid's Algorithm Analysis

At every iteration of the algorithm, the following statements are true:

$$0 \leq r_i < r_{i-1}$$

$$r_i = aX_i + bY_i$$

$$\gcd(a, b) \mid r_i$$

If these statements are true for i, the statements also hold true for i+1 (by the arguments before). They are true for i=0 and thus we prove by induction that the statements are true for all i.

Since r_i strictly decreases, the algorithm must eventually reach $r_i = 0$, at which point it terminates with $i - 1 = i_f$. At that point, $r_{i_f} \mid r_{i_f-1}$. But that means $r_{i_f} \mid r_{i_f-2} = m_{i_f-1} r_{i_f-1} + r_{i_f}$ and so on. By induction, we also have $r_{i_f} \mid r_j$ for all j.

Euclid's Algorithm Analysis

At every iteration of the algorithm, the following statements are true:

$$0 \leq r_i < r_{i-1}$$

$$r_i = aX_i + bY_i$$

$$\gcd(a, b) \mid r_i$$

If these statements are true for i, the statements also hold true for i+1 (by the arguments before). They are true for i=0 and thus we prove by induction that the statements are true for all i.

Since r_i strictly decreases, the algorithm must eventually reach $r_i = 0$, at which point it terminates with $i - 1 = i_f$. At that point, $r_{i_f} \mid r_{i_f - 1}$. But that means $r_{i_f} \mid r_{i_f - 2} = m_{i_f - 1} r_{i_f - 1} + r_{i_f}$ and so on. By induction, we also have $r_{i_f} \mid r_j$ for all j.

In particular, $r_{i_f} \mid a$ and $r_{i_f} \mid b$. But $\gcd(a, b) \mid r_{i_f}$, so

$$r_{i_f} = \gcd(a, b)$$

Efficiency of Euclid's Algorithm

How quickly does r_i decrease in Euclid's algorithm?

If $r_i \geq r_{i-1}/2$, then $r_{i+1} \leq r_{i-1}/2$.

If $r_i \leq r_{i-1}/2$, then $r_{i+1} \leq r_i \leq r_{i-1}/2$.

Either way, $r_{i+1} \leq r_{i-1}/2$.

Since r_i is at least halved every 2 steps, the algorithm can run at most $2\log_2 a$ steps before halting.

Meaning of Efficient

It's important to remember that efficient (or polynomial time) means polynomial time as a function of the input size.

When doing arithmetic or finding the gcd, the input size is the length (i.e., number of bits) of the numbers being computed with.

Not polynomial in the numbers themselves!

Integer addition, subtraction, multiplication, division (with remainder) are all efficient in this sense using standard grade school algorithms. Still true for modular +, -, *.

$\log_2 a$ is the input size, so Euclid's algorithm has a polynomial number of steps, each of which is efficient. Therefore it is efficient overall.

Modular Division

If $\gcd(b, N) = 1$, then we can always divide by b in mod N arithmetic:

Using Euclid's algorithm, find X,Y such that

$$bX + NY = 1$$

Then $bX = 1 \bmod N$.

X is then the multiplicative inverse of b:

$$(aX)b = a(Xb) = a \bmod N$$

so $a/b = aX \bmod N$.

And moreover, we can divide in polynomial time.

Example: $1 = -5 \cdot 57 + 13 \cdot 22$

Thus, $a/22 \bmod 57 = 13a \bmod 57$. E.g., $5/22 = 8 \bmod 57$

Dos and Don'ts of Division

When b and N are relatively prime, it is OK to cancel b from an equation:

$$ab = cb \bmod N \Rightarrow a = c \bmod N$$

But this is *not* OK in general if $\gcd(b, N) \neq 1$.

Examples:

$2 \cdot 4 = 2 \cdot 9 \bmod 10$ but $4 \neq 9 \bmod 10$.

$$3 \cdot 4 + 3 \cdot 4 = 4 \bmod 10 = 3 \cdot 8 \bmod 10$$

$$\Rightarrow 4 + 4 = 8 \bmod 10$$

Diffie-Hellman Security Idea

In Diffie-Hellman, Alice and Bob must perform modular exponentiation: Alice announces $A = g^a \bmod p$ and Bob announces $B = g^b \bmod p$ for secret a and b chosen by Alice and Bob respectively and not shared with each other or Eve. Then they do another pair of modular exponentiations B^a and A^b to calculate the key.

- Therefore, Alice and Bob need efficient algorithms to compute modular exponentials.

Eve can break Diffie-Hellman if she can calculate the discrete log for (g,p): That is, if given y, she can find x such that $g^x = y \bmod p$.

- So, for security, we need that calculating the discrete log is hard.

Efficiency of Modular Exponentiation

In order to run Diffie-Hellman, we need to perform modular exponentiation. Can we do this efficiently as a function of the length of the numbers involved?

To calculate $g^a \bmod p$, we could:

- Start with $g \bmod p$.
- Multiply by g a total of a times, each time reducing mod p after the multiplication.

However, this takes a total of a multiplications, which is too many: $a = O(\exp(\log a))$.

Since Eve can also find the discrete log in O(a) multiplications by computing all the powers of g, we definitely need a better algorithm for modular exponentiation.

Repeated Squaring

We can get large exponents quickly by repeated squaring:

From $g^i \bmod p$, we can calculate $g^{2i} \bmod p$ using 1 multiplication by squaring it.

Doing this repeatedly gives us $g, g^2, g^4, g^8, \ldots, g^{2^c}$, with only c multiplications.

To calculate $g^a \bmod p$ for general a, first write a in binary:

$$a = a_0 2^c + a_1 2^{c-1} + \cdots + a_{c-1} 2 + a_c$$

Then $$g^a = \prod_{i=0}^{c} g^{a_{c-i} 2^i}$$

This needs $O(\log a)$ multiplications.

This class is being recorded

Example:

Calculate $65^{12} \bmod 71$:

$$65^2 = 36 \bmod 71$$

$$65^4 = 36^2 = 18 \bmod 71$$

$$65^8 = 18^2 = 40 \bmod 71$$

Then

$$\begin{aligned} 65^{12} &= 65^8 \cdot 65^4 \bmod 71 \\ &= 40 \cdot 18 \bmod 71 \\ &= 10 \bmod 71 \end{aligned}$$

How Many Powers Are There?

For discrete log to be hard, we need for g^a to take on many different possible values for fixed g.

How many can it take? The answer depends on both g and p.

Since there are only p-1 possible values mod p, eventually g^a must repeat, $g^{r+1} = g \bmod p$. Let us assume g and p are relatively prime so we can cancel g. Then $g^r = 1 \bmod p$.

> Definition: If r is the lowest power for which $g^r = 1 \bmod p$, then r is the order of g, ord(g).

After r, powers of g start to repeat:

$$g^a = g^{\mathrm{ord}(g)} g^{a-\mathrm{ord}(g)} = 1 \cdot g^{a-\mathrm{ord}(g)} = g^{a-\mathrm{ord}(g)} \bmod p$$

Or more generally,

$$g^a = g^b \bmod p \text{ iff } a = b \bmod \mathrm{ord}(g)$$

This class is being recorded

Modular Exponentiation Example 1

Mod 10: We will focus only on g which are relatively prime to 10.

$$3^1 = 3 \bmod 10$$
$$3^2 = 9 \bmod 10$$
$$3^3 = 7 \bmod 10$$
$$3^4 = 1 \bmod 10$$

ord(3) = 4

$$7^1 = 7 \bmod 10$$
$$7^2 = 9 \bmod 10$$
$$7^3 = 3 \bmod 10$$
$$7^4 = 1 \bmod 10$$

ord(7) = 4

$$9^1 = 9 \bmod 10$$
$$9^2 = 1 \bmod 10$$

ord(9) = 2

Notice that all numbers relatively prime to 10 appear as exponents of 3 and 7.

Modular Exponentiation Example 2

Mod 11: Now every g is relatively prime to 11.

$3^1 = 3 \bmod 11$

$3^2 = 9 \bmod 11$

$3^3 = 5 \bmod 11$

$3^4 = 4 \bmod 11$

$3^5 = 1 \bmod 11$

ord(3) = 5

The order can be much higher mod 11 than mod 10.

$7^1 = 7 \bmod 11$

$7^2 = 5 \bmod 11$

$7^3 = 2 \bmod 11$

$7^4 = 3 \bmod 11$

$7^5 = 10 \bmod 11$

$7^6 = 4 \bmod 11$

$7^7 = 6 \bmod 11$

$7^8 = 9 \bmod 11$

$7^9 = 8 \bmod 11$

$7^{10} = 1 \bmod 11$

ord(7) = 10

Exponentiation and GCD

Proposition: If $\gcd(g,p) = 1$, then $\gcd(g^a \bmod p, p) = 1$ as well:

Proof:

We can assume $a < r = \text{ord}(g)$. Then

$$g^a g^{r-a} = g^a = 1 \bmod p$$

But this implies that g^{r-a} is the multiplicative inverse of g^a.

Notice that $\gcd(h,p) \mid hk$ for all k and $\gcd(h,p) \mid p$, so $\gcd(h,p) \mid (hk \bmod p)$. In particular, if $\gcd(h,p) \neq 1$, then there is no k such that $hk = 1 \bmod p$.

Since g^a has a multiplicative inverse, it follows that $\gcd(g^a \bmod p, p) = 1$.