# CMSC/Math 456: Cryptography (Fall 2022)

## Lecture 4
Daniel Gottesman

# Administrative

Remember the first problem set is due at noon next Tuesday, Sep. 13.

There was an error in problem 2b of the first problem set (now corrected): it asked you to calculate "the conditional distribution $\text{Prob}(M = m)$," which should have been $\text{Prob}(M = m \mid C = \text{horse})$.

And remember, if you are reading these slides before the lecture, stop and think when you get to the vote.

This class is being recorded

# Recap: Computational Limits

Recall that last time we discussed applying computational limits to Eve's attacks. We decided that we would usually insist that Eve's attack be completed in *polynomial time* as a function of the input size and security parameter.

> This means that Eve's attack uses a number of computational steps equal to f(n), where n is the size of the ciphertext (which itself usually scales with s, the security parameter), for $f(n) = O(n^c)$, for some constant c.

This gives us a well-defined theory, but has the caveat that for practical applications we can't just think about the limit of infinite n, but should plug in the actual parameters of our protocol to see what level of security we get in practice.

This class is being recorded

# Improving on the One-Time Pad

How can we use this approach to improve on the one-time pad?

We wish to replace the long key with a bit string that looks random to Eve but that can be generated in the same way by both Alice and Bob using a short shared key. This gives the pseudo one-time pad.

For our threat model, we will assume that:

- Eve knows the technique we are using to generate the bit string but doesn't know the short key.
- Eve can perform arbitrary polynomial-time computations.

What does "looks random" mean?

Does it mean that 0 and 1 are equally likely for each bit?

No. 50% chance of 000000 and 50% chance of 111111 has that property, but correlations matter too.

This class is being recorded

# An Example

Here are two sequences of 50 bits. One was generated by rolling dice and one was generated by repeatedly applying a deterministic procedure to a random starting point:

Sequence #1:

01101001000010111010110101001111111000011000001100

Sequence #2:

11011101110100001011111111011001110000111010001011O

Which is which? Vote.

This class is being recorded

# The Example Continued

The non-random sequence was generated by the rule

$$x_{i+1} = Ax_i + B \bmod 16$$

writing each $x_i$ in binary. A and B are secret and the starting point was also generated with dice.

Sequence #1:

0110100100001011101011010100111111110000110000011100

Sequence #2:

1101110111010000101111111011001110000111010001011010

Does anyone want to change their vote?

This class is being recorded

Consider $x_{i+1} = Ax_i + B$ mod 16.

- If A is even, then $x_i$ always has the same parity as B, and is therefore either always even or always odd (except the first one). In either case, every 4th bit is the same ($i = 0$ mod 4).
- If A is odd and B is even, then $Ax_i$ has the same parity as $x_i$, and therefore $x_{i+1}$ also has the same parity as $x_i$. Again, every 4th bit is the same.
- If A is odd and B is odd, then $x_{i+1}$ will have the *opposite* parity from $x_i$. Now every 4th bit alternates 0 and 1.

So for the non-random sequence, look at every 4th bit. Either they are always the same (except maybe the first one) or they alternate 0 and 1. The random sequence doesn't have to follow either pattern and probably won't.

This class is being recorded

# The Answer to the Example

$$x_{i+1} = Ax_i + B \bmod 16$$

Look at every 4th bit:

Sequence #1:

0110100100001011010110101001111111100001100000110 0

Sequence #2:

110111011101000010111111101100111000011101000101 10

Sequence #1 alternates.  Sequence #2 doesn't alternate or stay the same.  Sequence #2 is the random one.

The parameters I used were A=13, B=11.

- It's not always easy to tell the difference between random numbers and numbers generated by a deterministic process.
- But sometimes there are subtle ways to distinguish the two.

Does this matter for cryptography? Yes!

If you use the pseudo one-time pad with a string that has detectable patterns, Eve can potentially use those to attack the encryption. Remember, we want to be conservative. Even if we don't know how to use the pattern to break the encryption, can we be sure that Eve does not?

Built-in random generators in many old programming languages use a version of the same linear congruent generator in the example with better parameters. More modern languages, like Python, use the Mersenne twister. But both methods have subtle correlations that make them unsafe for cryptography.

This class is being recorded

However, we are on the right track. We need a procedure for which any patterns are so subtle, Eve can't find them within the limits of her computational power.
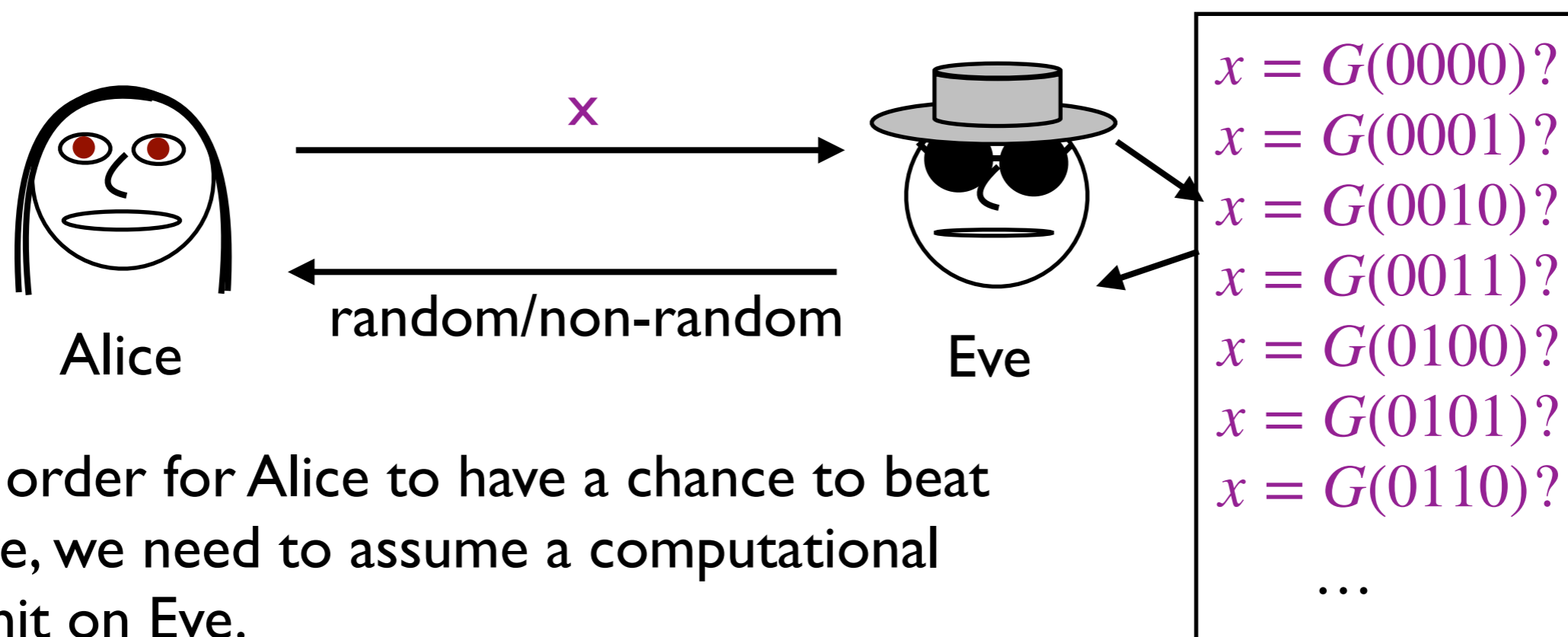
We can formalize this through a game of the same sort as we just tried. Alice presents Eve with a random string and then Eve must guess whether the string is truly random or was produced through the deterministic process. (Again, assuming Eve knows the process but not some specific parameters used as the key.)



x

random/non-random

Alice

Eve

Attack

This class is being recorded
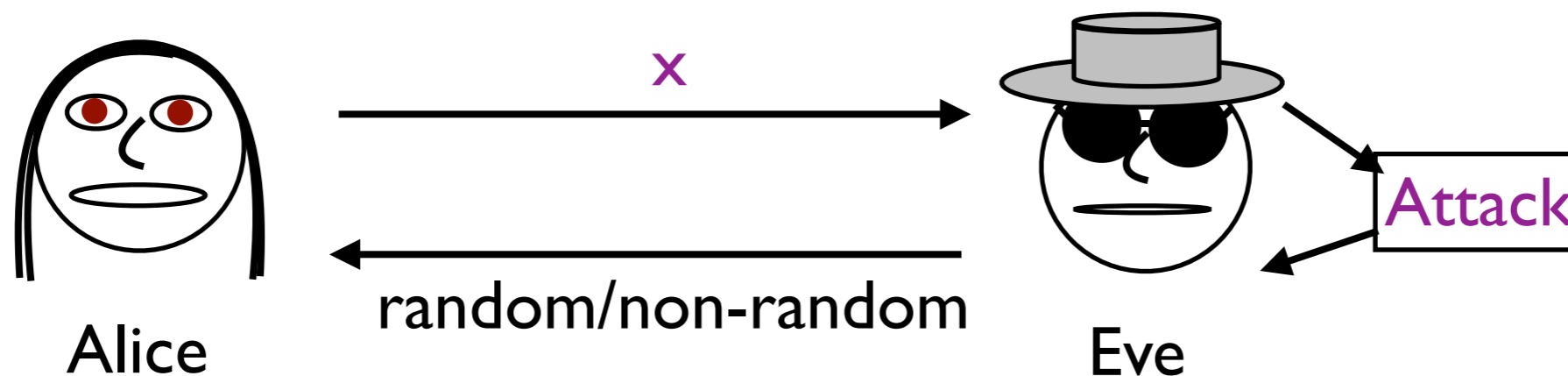
Alice's procedure is deterministic with a small random seed y. For each choice of y, there is a x = G(y) which Alice will use to challenge Eve (if she picks the deterministic procedure).

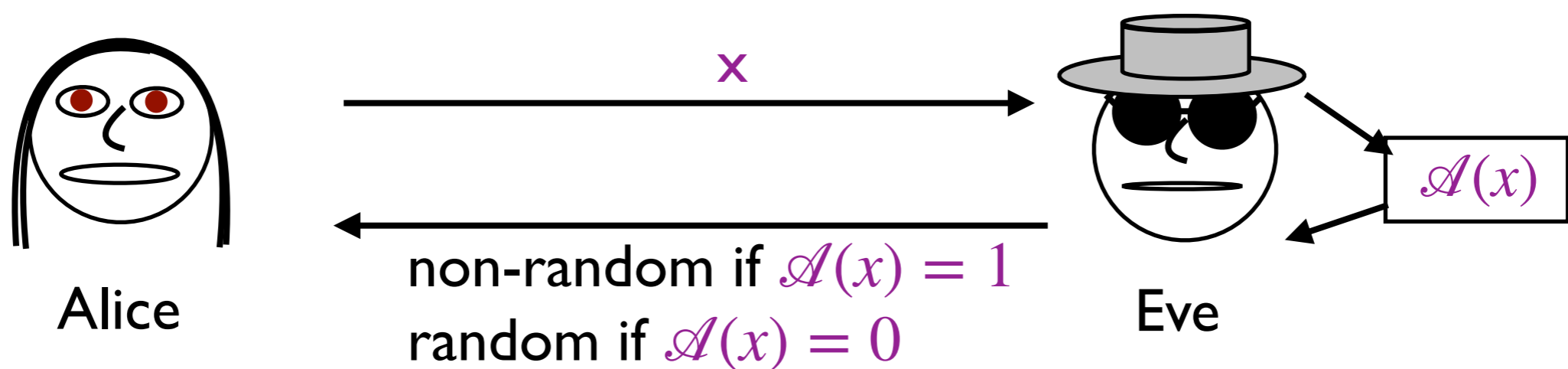But the set of possible x's that you can get this way is much smaller than the list of possible random numbers.

x

random/non-random

Alice

Eve

$x = G(0000)?$
$x = G(0001)?$
$x = G(0010)?$
$x = G(0011)?$
$x = G(0100)?$
$x = G(0101)?$
$x = G(0110)?$

…

In order for Alice to have a chance to beat Eve, we need to assume a computational limit on Eve.

# Pseudorandom Generators

Definition: Let $G : \{0,1\}^s \to \{0,1\}^{\ell(s)}$ be a *deterministic* function with $\ell(s) > s$ for all s. Then G(y) is a pseudorandom generator if, for any attack $\mathscr{A} : \{0,1\}^{\ell(s)} \to \{0,1\}$, a probabilistic polynomial time algorithm, it holds that

$$|\Pr_y(\mathscr{A}(G(y)) = 1) - \Pr_x(\mathscr{A}(x) = 1)| \leq \epsilon(s)$$

with $\epsilon(s)$ a negligible function and probabilities averaged over randomness of $\mathscr{A}$, as well as over seeds y (left probability) drawn uniformly from $\{0,1\}^s$ and truly random strings x (right probability) drawn uniformly from $\{0,1\}^{\ell(s)}$.



x

Alice

non-random if $\mathscr{A}(x) = 1$
random if $\mathscr{A}(x) = 0$

$\mathscr{A}(x)$

Eve

This class is being recorded

# Does Pseudorandomness Exist?

A pseudorandom generator outputs strings for which any polynomial-time algorithm can't successfully distinguish from random strings.

How can we build pseudorandom generators? Not straightforward, but we will discuss how to do it later.

To prove that these constructions give pseudorandom generators, we need to prove that there is no efficient way to distinguish the output from random bits.

This class is being recorded

# Does Pseudorandomness Exist?

A pseudorandom generator outputs strings for which any polynomial-time algorithm can't successfully distinguish from random strings.

How can we build pseudorandom generators? Not straightforward, but we will discuss how to do it later.

To prove that these constructions give pseudorandom generators, we need to prove that there is no efficient way to distinguish the output from random bits.

… but unfortunately, we don't know good ways of proving that something can't be done in polynomial time. If we could prove a pseudorandom generator exists, we could show P ≠ NP.

Instead, we try different kinds of attacks on the generator, and if they fail, we tentatively assume it works, with more confidence over time if it is still not broken.

This class is being recorded

# Cryptographic Primitives

Suppose now that we have something we think is a pseudorandom generator. We can then use it to create other cryptographic protocols. A pseudorandom generator is an example of a cryptographic primitive.

We can use cryptographic primitives as components in a larger (and maybe more useful) cryptographic protocol. For instance, we will use pseudorandom generators to create a more efficient encryption scheme, the pseudo one-time pad.

Pseudo one-time pad

Pseudorandom generator
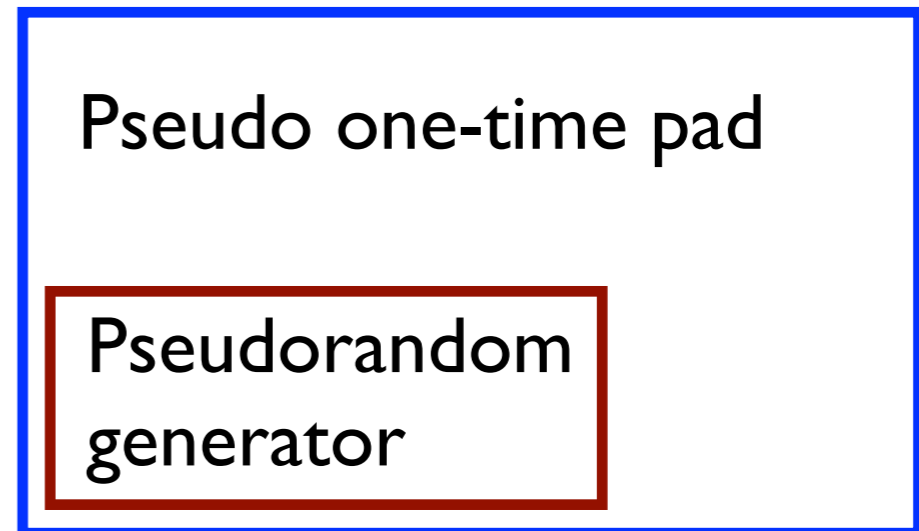
If done properly, the security of the overall protocol can be proven *provided* the primitives being used are secure. This is where the precise definitions are essential.

This modular approach is particularly useful if the same primitive can be used in many places.

This class is being recorded

# Pseudo One-Time Pad

One-time pad:

> The security parameter s should be chosen to be equal to the message length to be used.
>
> Gen: Choose uniformly random bit string k of length s.
>
> Enc: Acts on message m as $Enc(k, m) = m \oplus k$.
>
> Dec: Acts on ciphertext c as $Dec(k, c) = c \oplus k$.

Pseudo one-time pad:

> Let G(y) be a pseudorandom generator. The security parameter s should be chosen so that $\ell(s)$ is equal to the length of the message to be used.
>
> Gen: Choose uniformly random bit string k of length s.
>
> Enc: Acts on message m as $Enc(k, m) = m \oplus G(k)$.
>
> Dec: Acts on ciphertext c as $Dec(k, c) = c \oplus G(k)$.

This class is being recorded

# Why Is This Secure?

It is hard to distinguish the output of a pseudorandom generator from random numbers. Another way of saying this is anything you do with the output of a pseudorandom generator should give the same results as doing the same thing with a random string.

One-time pad

c

Attack

???

Eve

Failed pseudo one-time pad

c

Attack

m

Eve
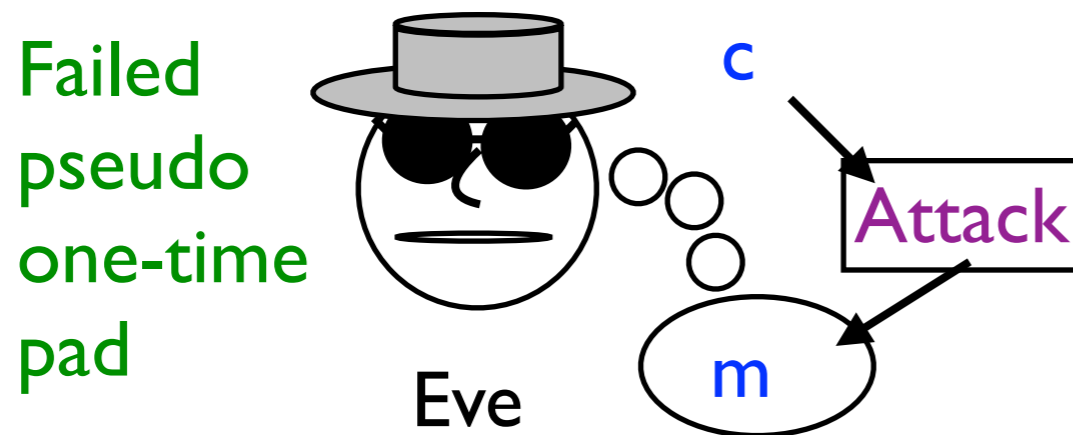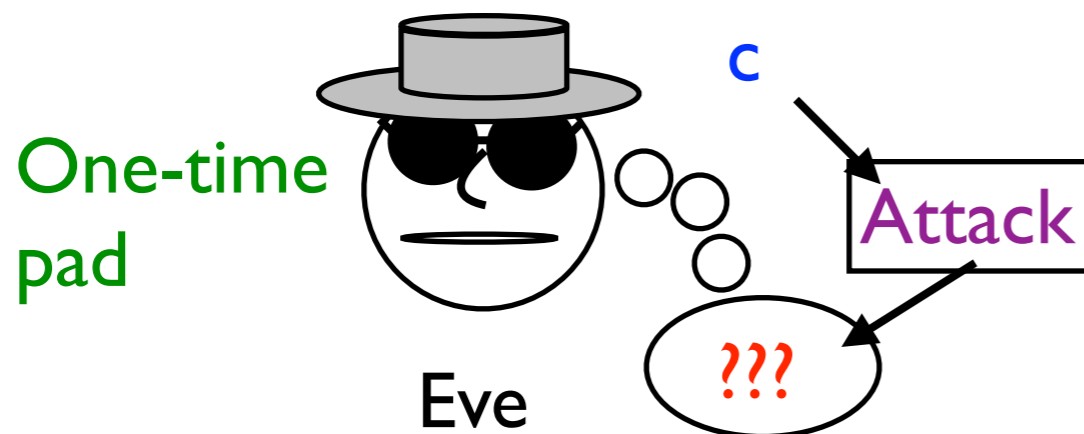
Eve cannot learn anything from a ciphertext encrypted with the one-time pad. If Eve has an attack on the pseudo one-time pad, then that is a different result than the one-time pad, which means the $G(y)$ you used could not have actually been a pseudorandom generator.

This class is being recorded

To actually prove this, we will need to devise a definition of security for encryption that allows computational limits on Eve.

We will start with definition B of perfect secrecy from last time and turn it into a game similar to the one we used to define pseudorandom generators.

Recall:

Definition B: An encryption protocol (Enc, Dec) provides perfect secrecy if for any pair of valid messages $m_0, m_1$, and any ciphertext c,

$$\Pr(Enc(k, m_0) = c) = \Pr(Enc(k, m_1) = c)$$

with probability averaged over keys k and randomness in Enc and Dec.

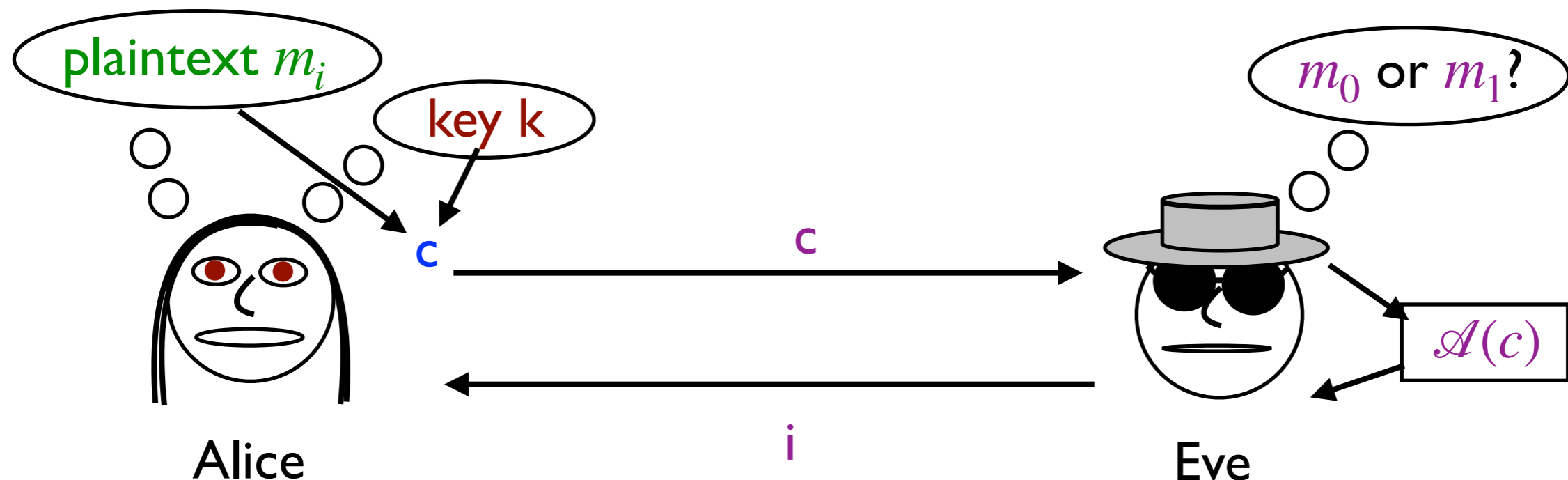In other words, Eve cannot guess if the message is $m_0$ or $m_1$.

This class is being recorded

Definition: (Enc, Dec) with security parameter s has indistinguishable encryptions in the presence of an eavesdropper (is EAV-secure) if, for any pair of messages $m_0$ and $m_1$ and for any efficient attack $\mathscr{A}(c)$,

$$|\Pr_k(\mathscr{A}(Enc(k, m_0)) = 1) - \Pr_k(\mathscr{A}(Enc(k, m_1) = 1)| \le \epsilon(s)$$

for negligible $\epsilon(s)$ and probability taken over k and randomness of Enc.



plaintext $m_i$

key k

$m_0$ or $m_1$?

c

c

$\mathscr{A}(c)$

i

Alice
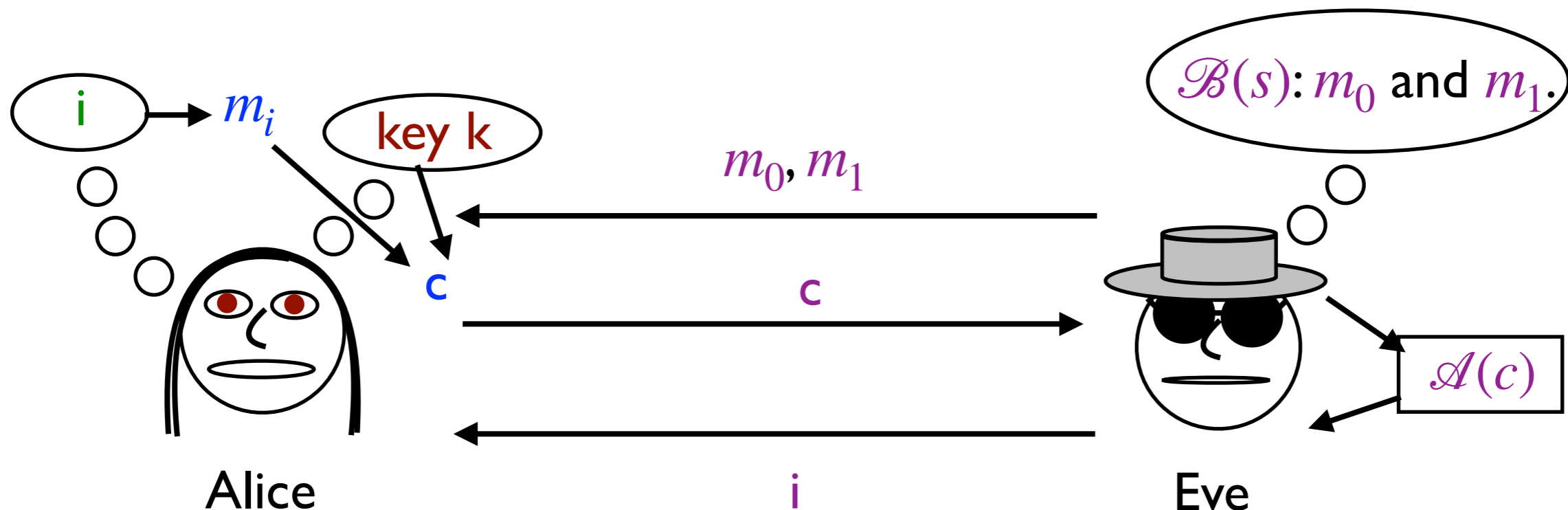
Eve

This class is being recorded

# EAV Security

Definition: (Enc, Dec) with security parameter s has indistinguishable encryptions in the presence of an eavesdropper (is EAV-secure) if, for any pair of messages $m_0$ and $m_1$ chosen by the adversary (using $\mathscr{B}(s)$) and for any efficient attack $\mathscr{A}(c)$,

$$|\Pr_k(\mathscr{A}(Enc(k, m_0)) = 1) - \Pr_k(\mathscr{A}(Enc(k, m_1)) = 1)| \leq \epsilon(s)$$

for negligible $\epsilon(s)$ and probability taken over k and randomness of Enc.



i $\quad m_i$ $\quad$ key k $\qquad m_0, m_1 \qquad \mathscr{B}(s): m_0$ and $m_1$.

c $\qquad$ c $\qquad \mathscr{A}(c)$

Alice $\qquad\qquad$ i $\qquad\qquad$ Eve

This class is being recorded

# Commentary on the Definition

How is does letting the adversary pick $m_0$ and $m_1$ change the definition? It actually makes the definition slightly weaker because Eve's algorithm $\mathscr{B}(s)$ to pick $m_0$ and $m_1$ is efficient.

Why did I do this?

- The weaker definition is easier to satisfy and it seems fine to say Eve might be able to distinguish a pair of messages that are hard to find.
- In later stronger definitions (such as the chosen plaintext attack), it will be more important to have Eve choose the possible messages.

Compared to the definition of perfect security, when we put the computational bound on Eve, we should also relax to negligible probability $\epsilon(s)$ of distinguishing the messages because Eve usually has a "guess the key" attack that succeeds with small probability.

This class is being recorded

# Proof By Reduction

Our intuition is that if Eve has an attack on the pseudo one-time pad, then it must have been that the function $G(y)$ used was not actually a pseudorandom generator. We can turn this into a proof by using the attack on the pseudo one-time pad to design an attack on the pseudorandom generator.

Attack on pseudorandom generator

Attack on pseudo one-time pad

If we are confident in the security of the pseudorandom generator, this turns around and implies that there can't be an effective attack on the pseudo one-time pad.

This is called a reduction: we are building the security of the pseudo one-time pad on that of the simpler cryptographic primitive used to create it. Reduction is an extremely common proof technique in cryptography and theoretical CS.

This class is being recorded

# Security of Pseudo One-Time Pad

Theorem: The pseudo one-time pad is EAV-secure if it uses a secure pseudorandom generator.
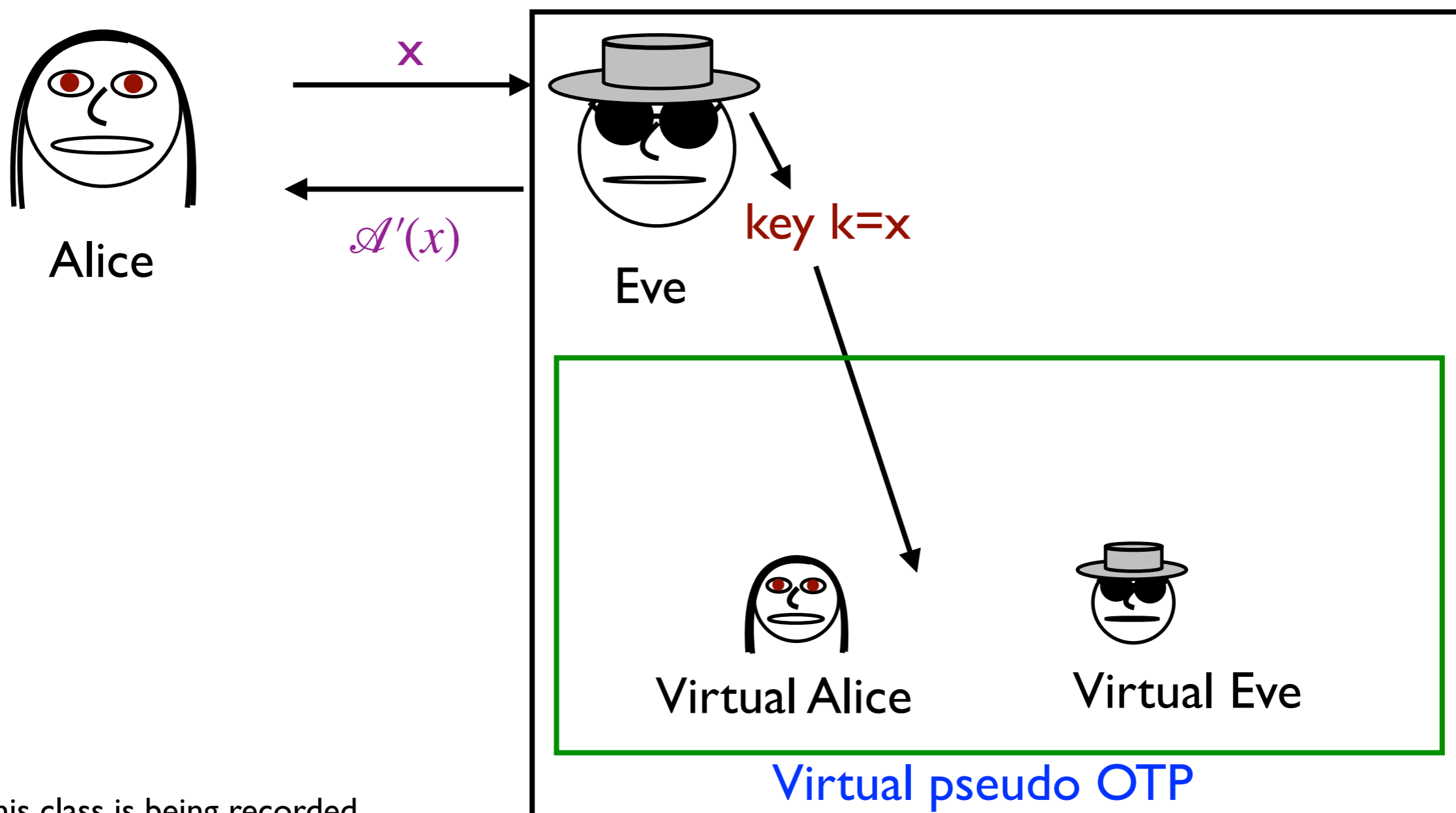
Proof plan:

Given an attack $(\mathcal{B}(s), \mathcal{A}(c))$ on the pseudo one-time pad, we will construct an attack $\mathcal{A}'(x)$ on the pseudorandom generator.

$\mathcal{A}'(x)$ will succeed in distinguishing the pseudorandom numbers from random with probability $\epsilon'(s)$ if $(\mathcal{B}(s), \mathcal{A}(c))$ succeeds in distinguishing messages with probability $\epsilon(s)$. If the pseudorandom generator is secure, $\epsilon'(s)$ will be negligible, which will imply that $\epsilon(s)$ is also negligible. Therefore this particular attack on the pseudo one-time pad doesn't succeed.

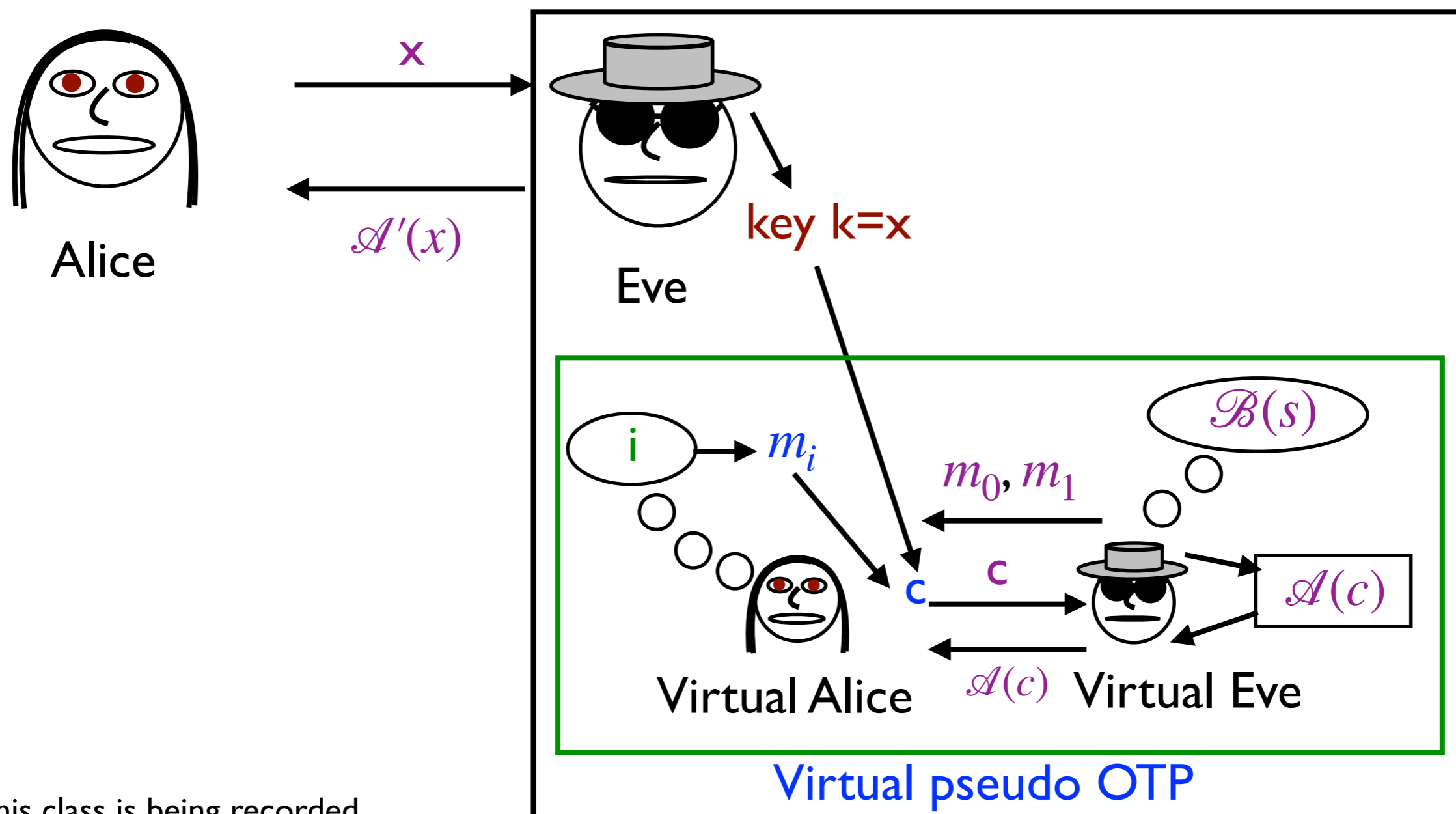But the attack was arbitrary, so the pseudo one-time pad is EAV-secure.

# The Reduction

The idea is that the attack $\mathscr{A}'(x)$ will use **x** as the key in a virtual pseudo one-time pad protocol, in which a virtual Eve will run the attack $(\mathscr{B}(s), \mathscr{A}(c))$ on the pseudo one-time pad.



x

$\mathscr{A}'(x)$

Alice

key k=x

Eve

Virtual Alice

Virtual Eve

Virtual pseudo OTP

This class is being recorded

# The Reduction

The idea is that the attack $\mathscr{A}'(x)$ will use **x** as the key in a virtual pseudo one-time pad protocol, in which a virtual Eve will run the attack $(\mathscr{B}(s),\mathscr{A}(c))$ on the pseudo one-time pad.
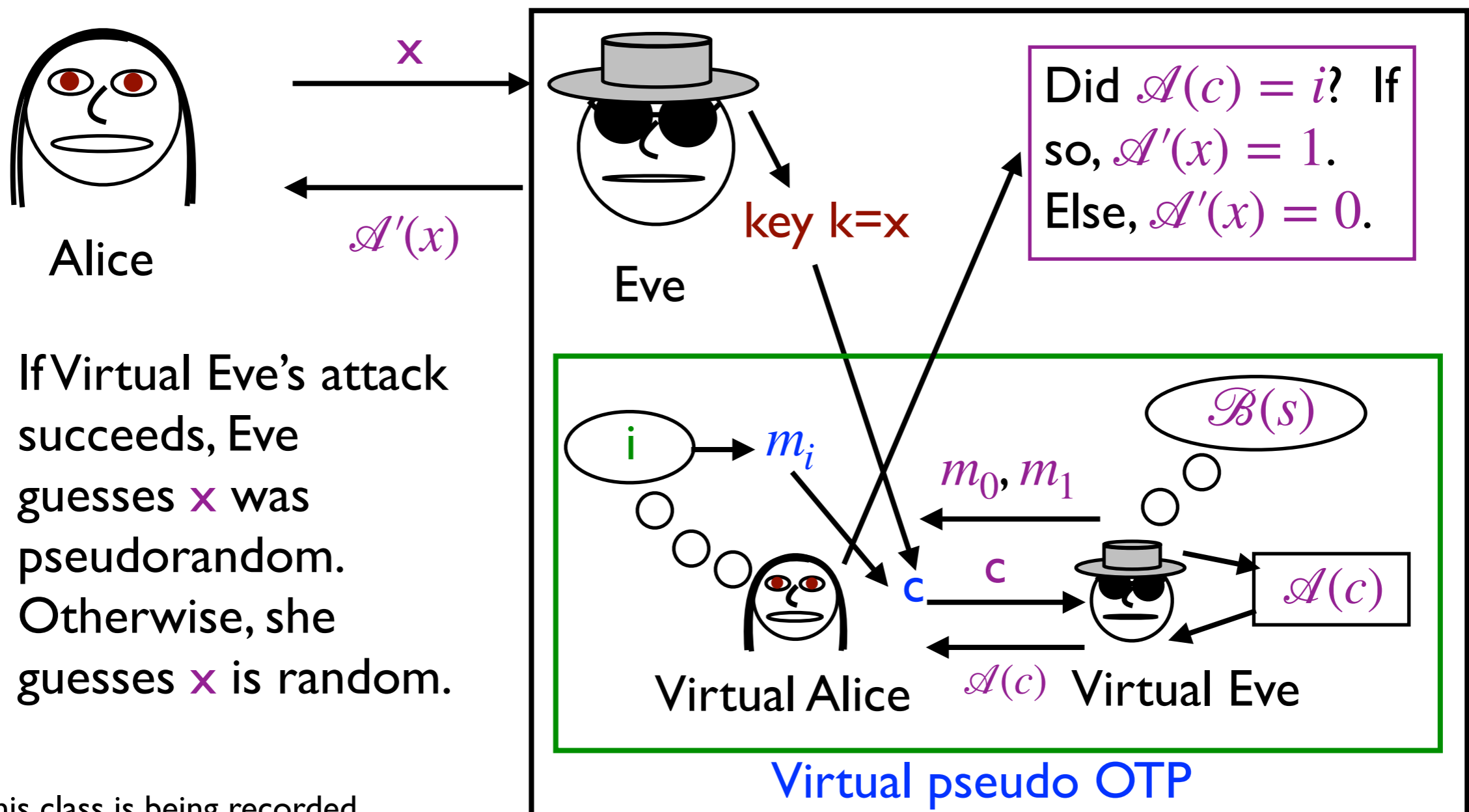
# The Reduction

The idea is that the attack $\mathscr{A}'(x)$ will use **x** as the key in a virtual pseudo one-time pad protocol, in which a virtual Eve will run the attack $(\mathscr{B}(s), \mathscr{A}(c))$ on the pseudo one-time pad.



If Virtual Eve's attack succeeds, Eve guesses **x** was pseudorandom. Otherwise, she guesses **x** is random.

This class is being recorded

# Why Does This Work?

The important thing to make this reduction work is that Virtual Eve has no idea she is just a simulation.  In particular, everything she sees is completely consistent with attacking either the one-time pad or the pseudo one-time pad.

> The only input and output to the virtual protocol goes through Alice, and Alice only makes choices she could have made in a real one-time pad or pseudo one-time pad.

Why is this important?  We want Virtual Eve to run her attack exactly as she would against a real encryption scheme.  If she were to change her attack, then we would only get results about her changed attack and not the true attack.  That would not help us prove security.

This class is being recorded

In the case where x was actually a uniform random string chosen by Alice, then the virtual protocol run was actually a virtual one-time pad. This means that it has perfect secrecy, and whatever pair of messages $(m_1, m_2)$ was chosen by $\mathcal{B}(s)$ and whichever ciphertext c ended up being used,

$$\Pr(Enc(x, m_1) = c) = \Pr(Enc(x, m_2) = c) = p_c$$

Now,

$$\Pr(\mathcal{A}(Enc(x, m_i)) = 1) = \sum_c \Pr(Enc(x, m_i) = c)\Pr(\mathcal{A}(c) = 1)$$

$$= \sum_c p_c\Pr(\mathcal{A}(c) = 1)$$

which doesn't depend on i. In particular,

$$\Pr(\mathcal{A}(Enc(x, m_1)) = 1) = \Pr(\mathcal{A}(Enc(x, m_2)) = 1)$$

This class is being recorded

To calculate the success probability for $\mathscr{A}'(x)$, first note that

$$\Pr(\mathscr{A}'(x) = 1) = \frac{1}{2}\left[\Pr(\mathscr{A}(Enc(x, m_1)) = 1) + 1 - \Pr(\mathscr{A}(Enc(x, m_0)) = 1)\right]$$

(Actually this should be averaged over pairs $(m_0, m_1)$ chosen by $\mathscr{B}(s)$, but we can specialize to $\mathscr{B}(s)$ with deterministic output.)

If $x$ is random, then $\Pr(\mathscr{A}'(x) = 1) = 1/2$.

This class is being recorded

In the case that $x = G(y)$ is pseudorandom, then let

$$\delta(x) = \Pr(\mathcal{A}(Enc(G(y), m_1)) = 1) - \Pr(\mathcal{A}(Enc(G(y), m_0)) = 1)$$

so $\Pr(\mathcal{A}'(G(y)) = 1) = 1/2\,(1 + \delta(G(y)))$ in this case. Note that here, probability is taken over randomness of Enc and $\mathcal{A}$, but we are not yet averaging over y.

When we do average over possible x and y values,

$$\Pr_y(\mathcal{A}'(G(y)) = 1) - \Pr_x(\mathcal{A}'(x) = 1) = \frac{1}{2}\sum_y \Pr(y)\delta(G(y))$$

so by the definition of a pseudorandom generator,

$$\left|\sum_y Pr(y)\delta(G(y))\right| \le 2\epsilon(s)$$

for negligible $\epsilon(s)$, which is also the formula needed for EAV security of the pseudo one-time pad.

This class is being recorded