CMSC/Math 456: Cryptography (Fall 2023)

Lecture 6

Daniel Gottesman

Administrative

Problem Set #3 is out. It is a Python programming assignment. The goal is to break a pseudorandom generator and pseudo one-time pad based on a poor stream cipher design.

> Note that a solution to break the pseudorandom generator does not automatically give you a solution to break the pseudo one-time pad, but it is likely to help a lot.

Problem Set #1 grades are available, and the solutions are available on ELMS as well.

> Regrade policy: Regrade requests are due within 1 week after the both the grades and the solution set are available.

This class is being recorded

Breaking RC4

Last time we saw a bias in the output stream of RC4; there are other biases and correlations as well. This means that for many uses, RC4 can be broken, even in practical situations.

In WEP, RC4 is modified to use an IV (to enable multiple messages with the same key, as we will discuss next) by substituting the public IV for the first 3 bytes of the key. This enables even stronger attacks, making WEP completely insecure.

Because of these various attacks, RC4 is no longer widely used.

Moral: Cryptography is hard!

While advances in computing power helped defeat RC4, primarily advances in *algorithms* and a better understanding of the structure of RC4 led to its defeat. Moreover, ad hoc modifications of cryptographic protocols are extremely dangerous.

This class is being recorded

Multiple Messages

One problem with unmodified RC4 and similar stream ciphers (ones without an IV) is that they always produce the same output when started with a given key k.

> This produces the same vulnerability as when the key for the one-time pad is used twice.

This means that if we want to use the same key repeatedly, something else has to change. This is where the IV comes in:

- IV could be a counter that increments with each message sent. This can work, but Alice and Bob must keep track how many messages were sent and if one gets lost, that causes problems.
- Or IV can be random each time and transmitted along with the ciphertext. This makes the ciphertext longer than the message.

This class is being recorded

Complications of Multiple Messages

To define security for an encryption protocol that is supposed to work for multiple messages, we need a new threat model. Presumably we should imagine that Eve has access to encryptions of more than one message.

> But what if Eve happens to know — or guess — the contents of one or more of those messages?
>
> We should include this possibility in the threat model. Sometimes the content of a message becomes obvious later, for instance if the message contains your plans for the day. At the end of the day, those plans have been revealed, but your encrypted plans for tomorrow are still relevant and an attractive target for Eve to break.

How can we formalize the idea that Eve knows some messages? Which messages and how similar or different might they be from the messages we are still trying to hide?

This class is being recorded

Chosen Plaintext Attack

Solution: Let Eve *pick* the plaintext of the other messages!

> We will assume Eve can pick some polynomial number of plaintext messages and see ciphertexts corresponding to those messages.

This is the conservative assumption. It is hard to quantify exactly which combinations of messages are likely, and also hard to quantify which ones will be most dangerous in terms of helping Eve break the protocol. But letting Eve pick the plaintexts guarantees the worst case, which means that all other cases where Eve knows some of the plaintexts are covered too.

> Obviously, I am not suggesting that Eve is telling Alice what to encrypt.

This class is being recorded

Chosen Plaintext Attack

Solution: Let Eve *pick* the plaintext of the other messages!

> We will assume Eve can pick some polynomial number of plaintext messages and see ciphertexts corresponding to those messages.

This is the conservative assumption. It is hard to quantify exactly which combinations of messages are likely, and also hard to quantify which ones will be most dangerous in terms of helping Eve break the protocol. But letting Eve pick the plaintexts guarantees the worst case, which means that all other cases where Eve knows some of the plaintexts are covered too.

> Obviously, I am not suggesting that Eve is telling Alice what to encrypt.
>
> Well …

This class is being recorded

Battle of Midway

During WW II, American cryptographers had partially broken the Japanese codes. In mid-1942, the U.S. intercepted communications indicating that Japan was planning a surprise attack against a U.S. target, but they were not initially able to decode the location "AF." They suspected that the target was Midway Island, but the stakes were high if they guessed wrong.

This class is being recorded

Battle of Midway

During WW II, American cryptographers had partially broken the Japanese codes. In mid-1942, the U.S. intercepted communications indicating that Japan was planning a surprise attack against a U.S. target, but they were not initially able to decode the location "AF." They suspected that the target was Midway Island, but the stakes were high if they guessed wrong.

> So they had the U.S. garrison on Midway send an unencrypted message saying they had a problem with their water distillation system and needed fresh water. And then they intercepted a Japanese message that decrypted to "AF is short of water."

The Battle of Midway is one of the major Pacific naval battles of WWII, and the U.S. victory is partially attributable to a chosen plaintext attack.

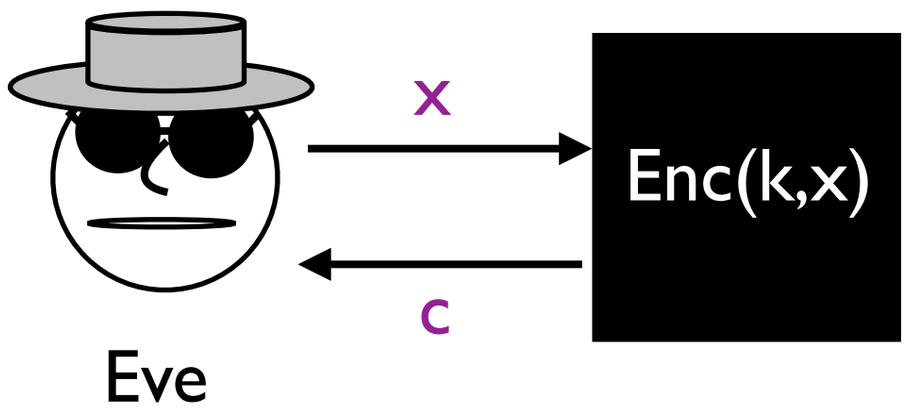
This class is being recorded

Oracles

So how do we define a chosen plaintext attack?

Recall that for EAV security, we used a game where Eve picked a pair of messages and then tried to guess which one Alice chose to encrypt.

> We need some way for Eve to pick the plaintext and get a ciphertext, but we don't want to give her the key. We will instead model this as an oracle, a black box to which Eve can ask questions but whose insides she cannot access.



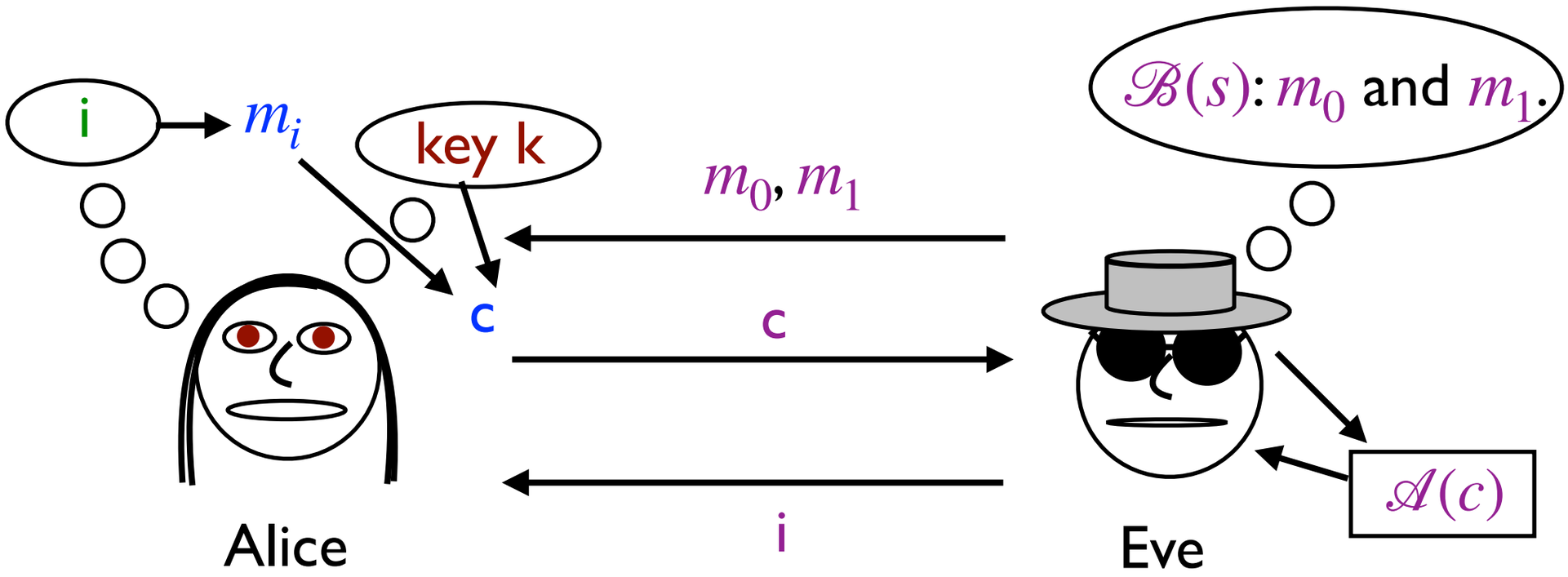
Eve can query this oracle up to polynomially many times at any time during the game.

This class is being recorded

CPA Security Game

Recall that for the EAV security game, there were three steps:

1. Eve chooses a pair of messages m_0 and m_1.
2. Alice picks one and encrypts it as ciphertext c.
3. Eve tries to guess which message c corresponds to.

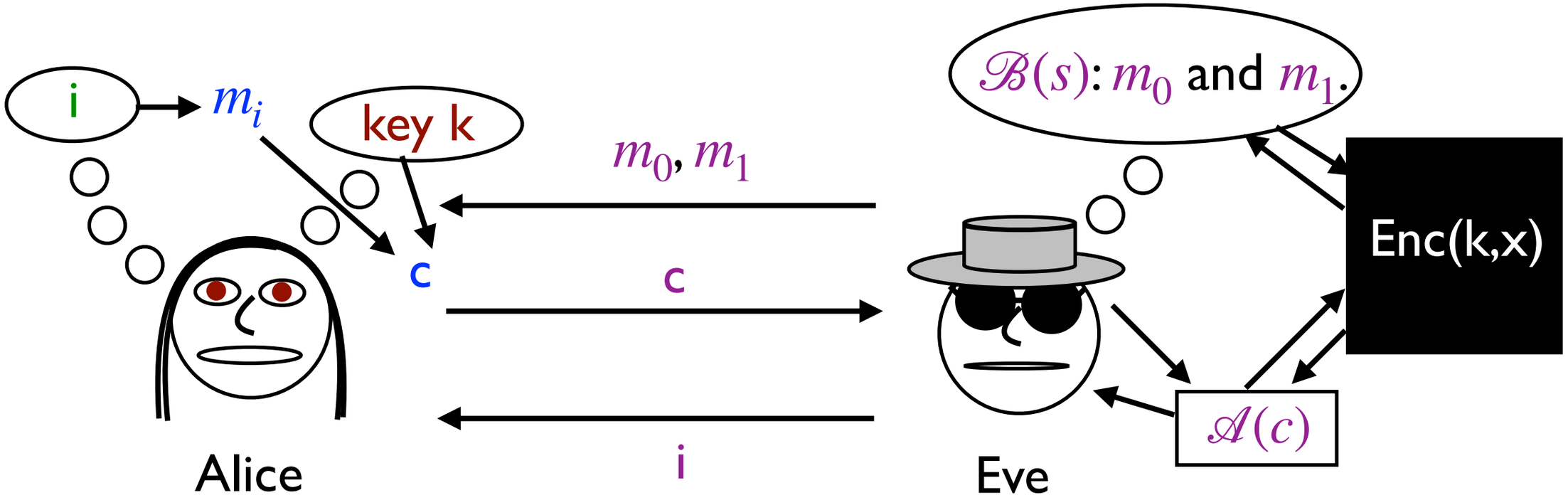


This class is being recorded

CPA Security Game

Recall that for the EAV security game, there were three steps:

1. Eve chooses a pair of messages m_0 and m_1.
2. Alice picks one and encrypts it as ciphertext c.
3. Eve tries to guess which message c corresponds to.



The CPA security game is the same except that Eve has access to the encryption oracle during both $\mathscr{B}(s)$ and $\mathscr{A}(c)$.

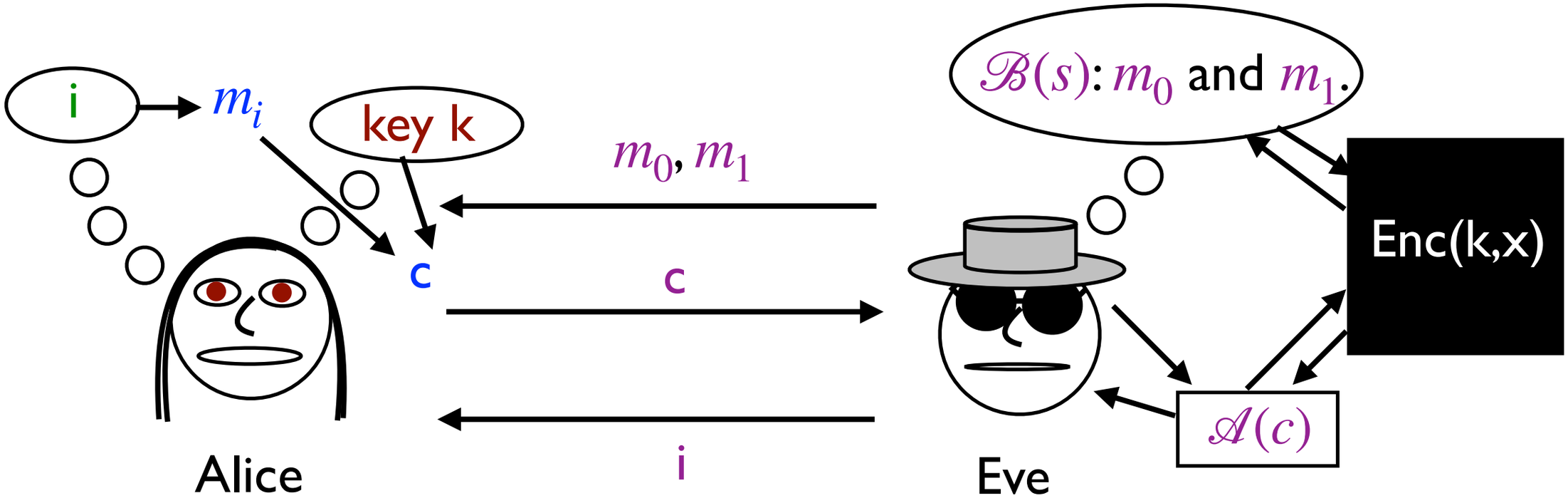
This class is being recorded

Definition of CPA Security

Definition: (Enc, Dec) with security parameter s is CPA-secure if, for any pair of messages m_0 and m_1 chosen by the adversary (using $\mathcal{B}(s)$ and oracle access to Enc(k,x)) and for any efficient attack $\mathcal{A}(c)$ (also with oracle access to Enc(k,x))

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

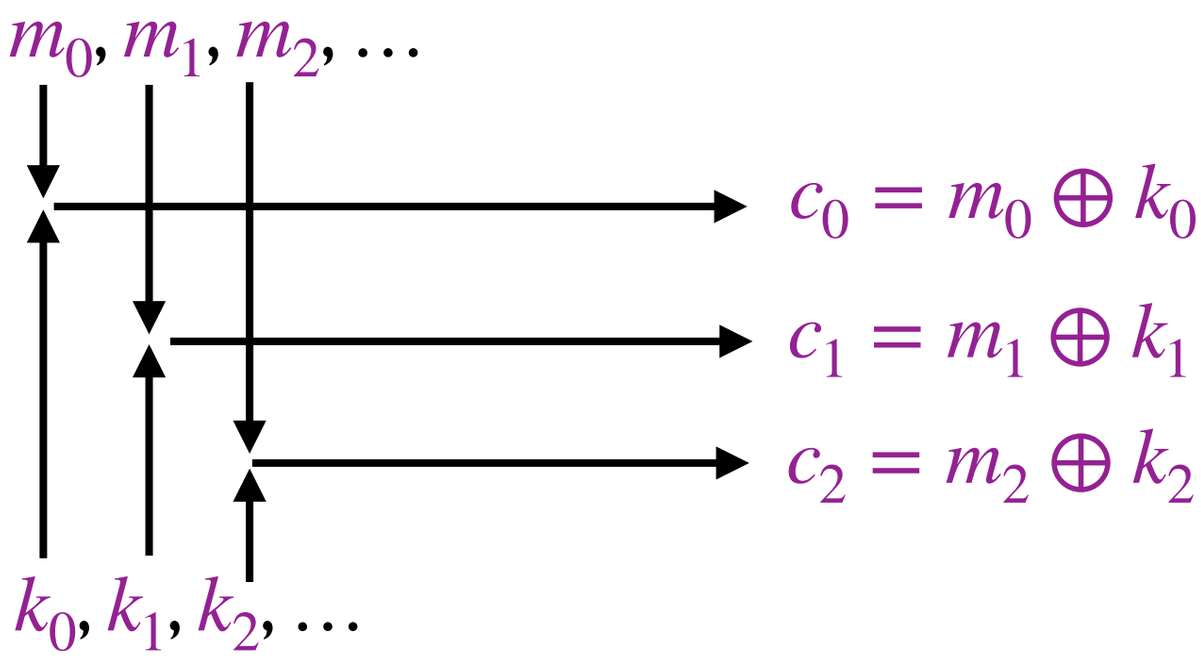
for negligible $\epsilon(s)$ and probability taken over k and randomness of Enc.



This class is being recorded

Achieving CPA Security

In order to achieve CPA security, we need to make sure the ciphertext for the current message is in some way independent of the previous messages. For instance, using the one-time pad, we could have a list of keys and use a different key for each message sent.



The first message sent or requested by Eve uses the first key k_0, the second message uses k_1, and so forth.

The messages being requested by Eve don't help her at all: the next key k_i to be used is unrelated to the previous ones.

This class is being recorded

Random Functions

We can view this strategy using different terminology as a *random function* indexed by a key k = $(k_0, k_1, k_2, \ldots)$:

$$F_k(r) = k_r$$

The function simply looks at the input r, picks the rth block of the key k, and outputs it.

Using r in order 0, 1, 2, … works here, but it runs a risk that if Alice and Bob fall out of sync, then Bob will no longer be able to decode Alice's messages.

Notice that for using this with the one-time pad, it doesn't actually matter that we use r in order, just that we don't repeat r's between different messages.

In fact, if we have a long enough list, it is good enough to pick a random r each time and announce it (so that Bob knows it).

This class is being recorded

Birthday Paradox

How many messages can we send before we are likely to see a random r repeated?

This can be analyzed using the "birthday paradox": There are about 25 people in this class. Even though there are many more days than that in the year, the odds are high that two of us have the same birthday.

Let's go around and compare.

This class is being recorded

Birthday Paradox

How many messages can we send before we are likely to see a random r repeated?

This can be analyzed using the "birthday paradox": There are about 25 people in this class. Even though there are many more days than that in the year, the odds are high that two of us have the same birthday.

Let's go around and compare.

The point is that with t people and N possible days for birthdays, there are $\binom{t}{2} = t(t-1)/2$ pairs of people, and each pair has probability 1/N of being a match, so we start to get matches when $t \approx \sqrt{2N}$ (which for N = 365 gives t in the mid-20s).

This class is being recorded

CPA Security With Random Functions

Let $F_k(r)$ be a secret family of random functions. Then the following protocol is CPA-secure by the previous argument, provided the number of messages is much less than $2^{s/2}$.

> Enc: Takes as input key k (of length s) and message m (of length n). Choose random r (of length n) and output ciphertext $c = (r, F_k(r) \oplus m)$.
>
> Dec: Takes as input key k and ciphertext c = (r, q). Outputs $m' := q \oplus F_k(r)$.

There are two problems with this protocol, however:

- It relies on $F_k(r)$ be secret, which seems like a violation of Kerckhoff's Principle. (Alternatively, you can have all of $F_k(r)$ as the key.)
- A full description of $F_k(r)$ is excessively long.

This class is being recorded

Pseudorandom Functions Motivation

If we want to choose a random r, we should therefore have a large pool to choose from. Say r is an n-bit number. Then we have 2^n possible values of r, so we should be OK up to about $2^{n/2}$ messages. That seems safe.

> Of course, if we do this with a truly random function $F_k(r)$, it is incredibly wasteful: the key k will need to be 2^n times the length of each message, and most values of r will never be used.

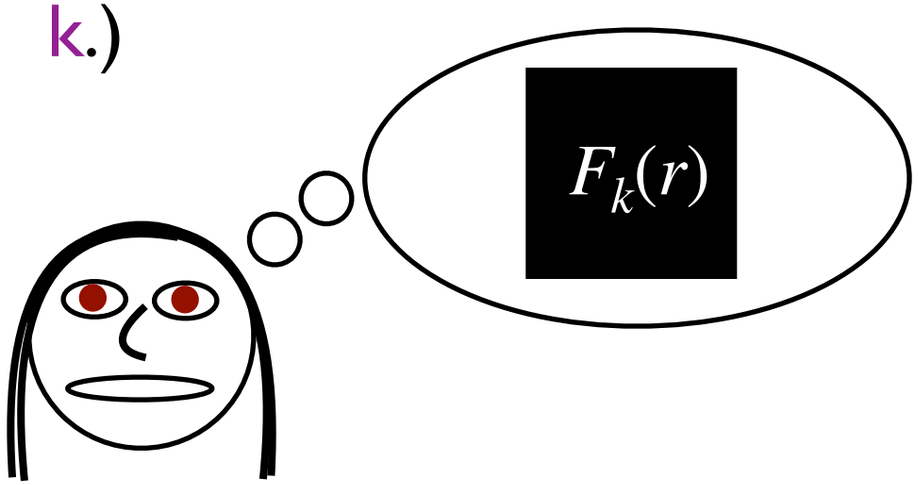
But if we can come up with a *pseudorandom function* $F_k(r)$ that is hard to distinguish from a random function, perhaps we can do this with a much smaller key.

This class is being recorded

Pseudorandom Functions

To define pseudorandom functions, we want to play another distinguishing game. We have to give Eve somehow a "copy" of $F_k(r)$ and ask her to determine if it is from a uniformly random ensemble of functions or from the pseudorandom ensemble.

A list of all possible input-output pairs would be far too long, so instead we imagine Alice constructing an oracle for $F_k(r)$, a black box Eve can query using inputs r but which she cannot look inside. (And in particular, she doesn't know k.)

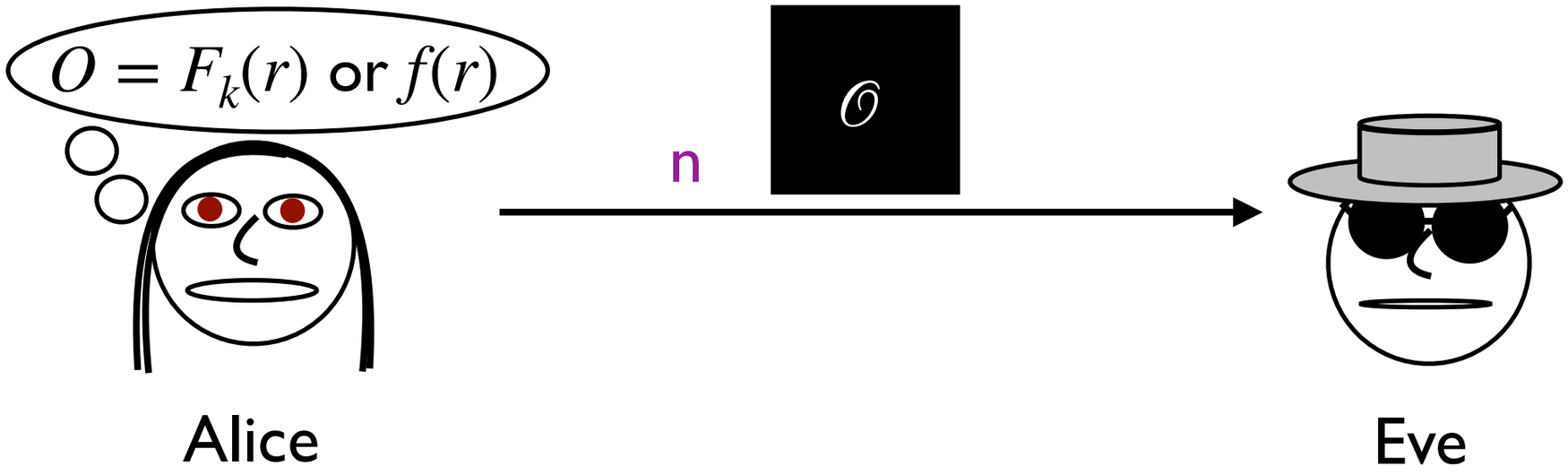


Alice

For simplicity and to make sure everything is polynomial, assume $|F_k(r)| = |r| = n$ and $|k| = s = \text{poly}(n)$.

This class is being recorded

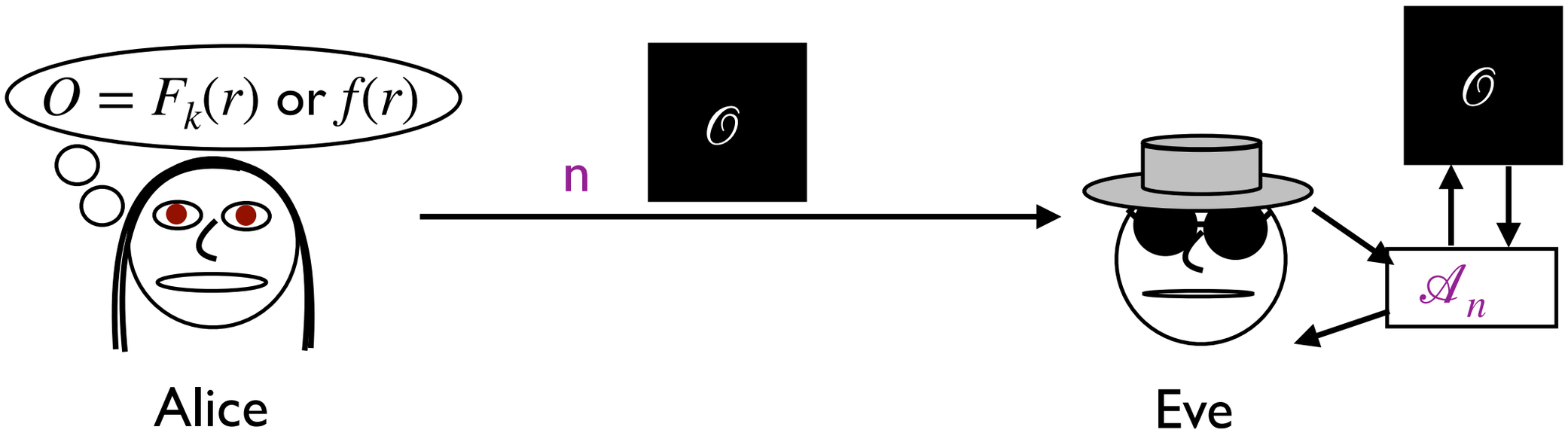
Pseudorandom Function Game



1. First, Alice prepares a block box $\mathcal{O}$, which takes as input r and computes either $F_k(r)$ for a random secret k (but always the same one each time $\mathcal{O}$ is used) or f(r) for a random function f (but again, always the same one each time $\mathcal{O}$ is used). She gives Eve access to $\mathcal{O}$.

This class is being recorded

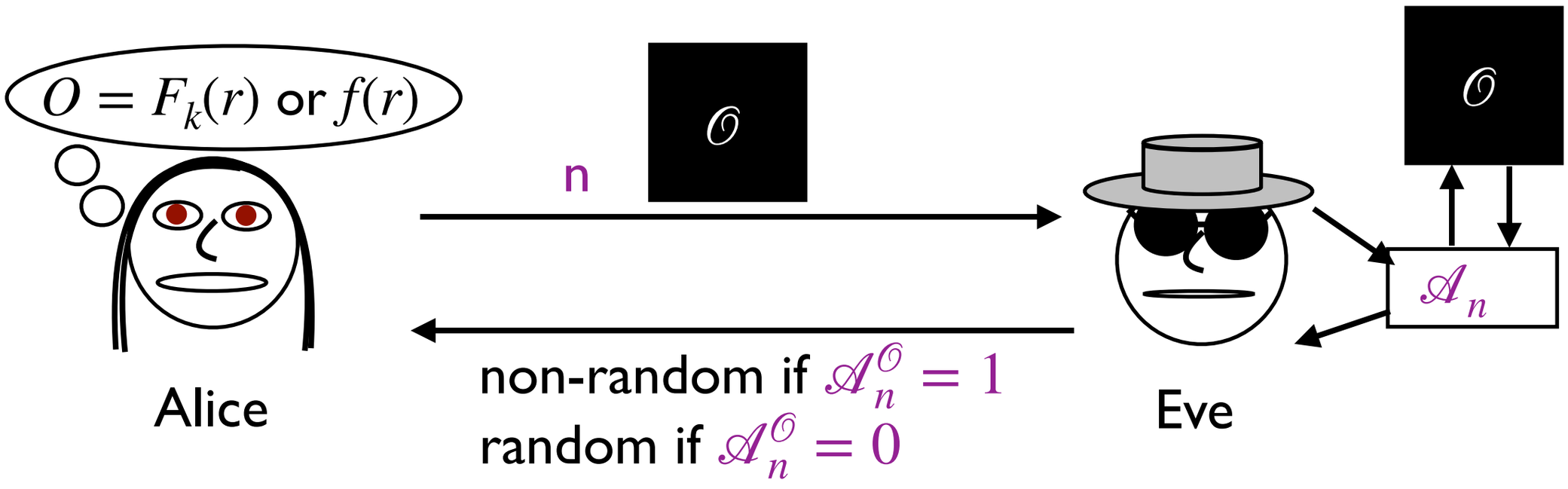
Pseudorandom Function Game



1. First, Alice prepares a block box $\mathcal{O}$, which takes as input r and computes either $F_k(r)$ for a random secret k (but always the same one each time $\mathcal{O}$ is used) or f(r) for a random function f (but again, always the same one each time $\mathcal{O}$ is used). She gives Eve access to $\mathcal{O}$.
2. Eve performs some attack using her access to $\mathcal{O}$.

This class is being recorded

Pseudorandom Function Game



1. First, Alice prepares a block box $\mathcal{O}$, which takes as input r and computes either $F_k(r)$ for a random secret k (but always the same one each time $\mathcal{O}$ is used) or f(r) for a random function f (but again, always the same one each time $\mathcal{O}$ is used). She gives Eve access to $\mathcal{O}$.
2. Eve performs some attack using her access to $\mathcal{O}$.
3. Eve returns her guess as to whether $\mathcal{O}$ computes F_k or f.

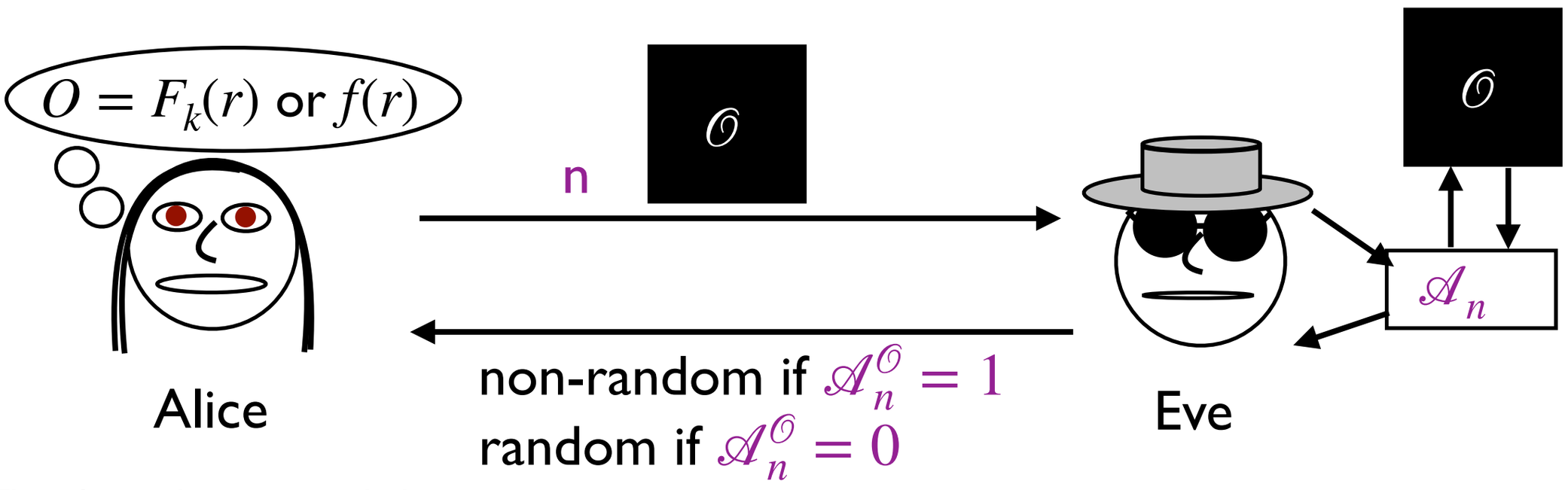
This class is being recorded

Pseudorandom Functions

Definition: Let $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be a *deterministic efficiently computable* function with $|F_k(r)| = |r| = n$ and $|k| = s = \text{poly}(n)$. Then $F_k(r)$ is a pseudorandom function if, for any efficient attack $\mathcal{A}_n$ that accesses an oracle and outputs a bit,

$$|\Pr_k(\mathcal{A}_n^{F_k(r)} = 1) - \Pr_f(\mathcal{A}_n^f = 1)| \leq \epsilon(s)$$

with $\epsilon(s)$ a negligible function and probabilities averaged over randomness of $\mathcal{A}$, as well as over uniformly random keys k (left probability) and truly random functions f (right probability).



This class is being recorded

Pseudorandom Permutations

For some applications, it is better to consider pseudorandom permutations rather than arbitrary pseudorandom functions. Consider $F_k(r)$ which is, for each k, a permutation, meaning it is invertible. Because it is a permutation, $|F_k(r)| = |r| = n$. We require that $F_k(r)$ and $F_k^{-1}(r)$ be efficiently computable and $|k| = s = \text{poly}(n)$. Such an $F_k(r)$ is a pseudorandom permutation if it is computationally indistinguishable from a random permutation.

Note that a pseudorandom permutation is also a pseudorandom function by the Birthday Paradox: Eve would need $O(2^{n/2})$ queries to notice that outputs are not repeating.

Why use them? For instance, pseudorandom permutations can be used to directly encrypt (ECB mode), i.e., ciphertext $c = F_k(m)$ since we can apply F_k^{-1} to decrypt.

This class is being recorded

Breaking ECB Mode

ECB mode: ciphertext is $c = F_k(m)$.
This is not CPA-secure because if Eve wants to decide between m_0 and m_1, she can just query the oracle on those two messages and compare the ciphertext with the two outputs. More generally, Eve can tell if two messages are the same or not.

Let us look at an example ECB ciphertext and use this fact to break the protocol:

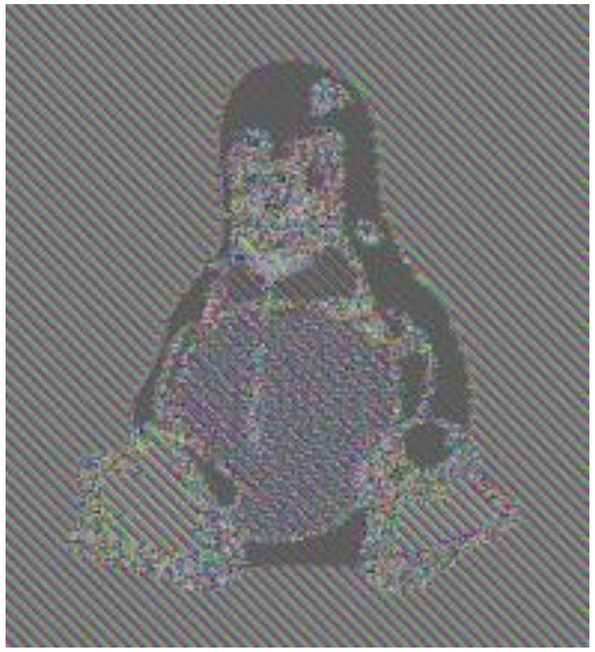
This class is being recorded

Breaking ECB Mode

ECB mode: ciphertext is $c = F_k(m)$.

This is not CPA-secure because if Eve wants to decide between m_0 and m_1, she can just query the oracle on those two messages and compare the ciphertext with the two outputs. More generally, Eve can tell if two messages are the same or not.

Let us look at an example ECB ciphertext and use this fact to break the protocol:



From Wikipedia, copyright Larry Ewing and User:Lunkwill

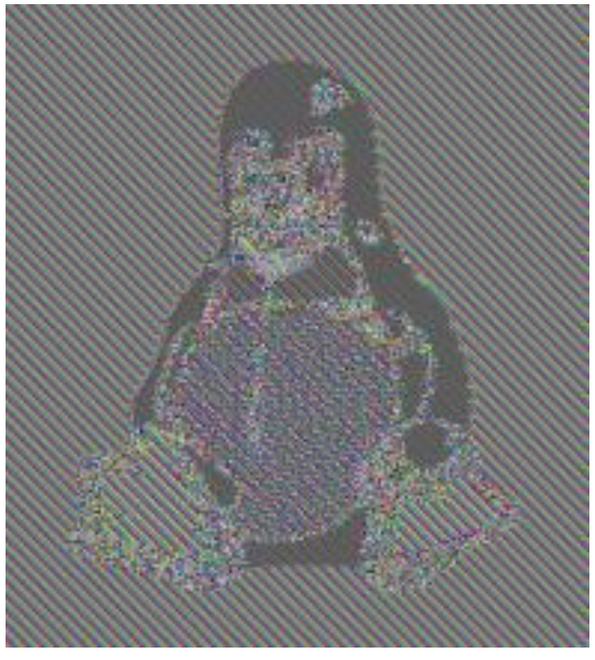
This class is being recorded

Breaking ECB Mode

ECB mode: ciphertext is $c = F_k(m)$.

This is not CPA-secure because if Eve wants to decide between m_0 and m_1, she can just query the oracle on those two messages and compare the ciphertext with the two outputs. More generally, Eve can tell if two messages are the same or not.

Let us look at an example ECB ciphertext and use this fact to break the protocol:



From Wikipedia, copyright Larry Ewing and User:Lunkwill

Decrypts to Tux Linux mascot by visual inspection.

This class is being recorded

Pseudorandomness and CPA Security

In order to avoid this problem, we need randomness in the encoding. For instance, with a pseudorandom function $F_k(r)$:

> Enc: Takes as input key k (of length s) and message m (of length n). Choose random r (of length n) and output ciphertext $c = (r, F_k(r) \oplus m)$.
>
> Dec: Takes as input key k and ciphertext c = (r, q). Outputs $m' := q \oplus F_k(r)$.
>
> Correctness is straightforward. Soundness (CPA-security) follows from the arguments we have made plus a reduction similar to the security proof for the pseudo one-time pad: The chance of repeating a random r is small. Therefore, if $F_k(r)$ is a truly random function, the protocol is secure. But a pseudorandom function should give the same results as a random function and we can show this via a reduction.

This class is being recorded

Block Cipher CTR Mode

In practice, practical constructions of pseudorandom functions and permutations have a fixed size. This means that we need some way of expanding them to longer messages.

One option: break the message up into pieces of size n and run the protocol from the last slide with a new random r for each block. Or we can save on randomness and length of the ciphertext by using a counter for the blocks. But we also need a random component to get security for multiple messages:

> Enc: Input key k, message $m = (m_1||m_2||m_3||\ldots)$. Choose random IV. Output ciphertext c = (IV, $m_1 \oplus F_k(IV||1)$, $m_2 \oplus F_k(IV||2), \ldots)$.

(Here || denotes concatenation.)

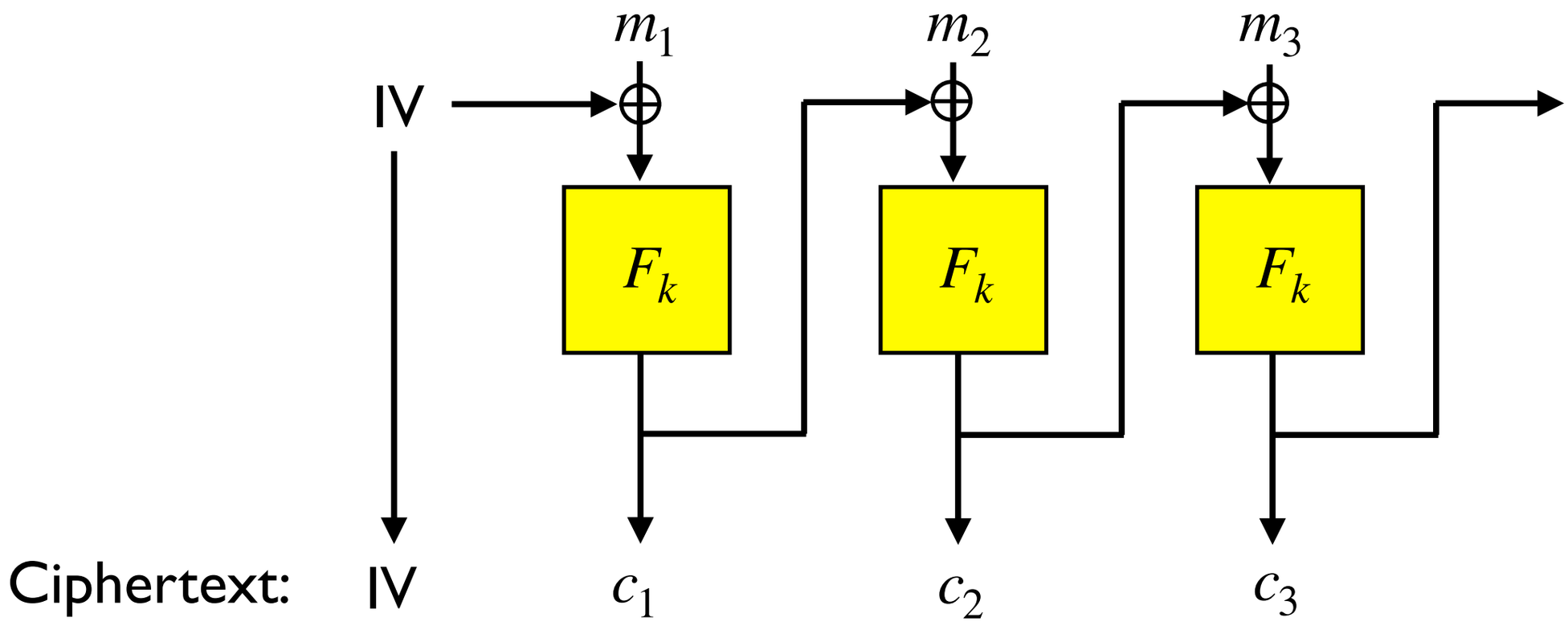
This is also CPA-secure since r=(IV||i) is unlikely to repeat within or between messages.

This class is being recorded

Block Cipher CBC Mode

Another popular option is known as CBC mode. It requires F_k which is a pseudorandom permutation. It is CPA-secure.

As with CTR mode, break the message up into blocks of size n and use a single IV.



To decrypt, Bob must invert F_k on block i and XOR with c_{i-1}.

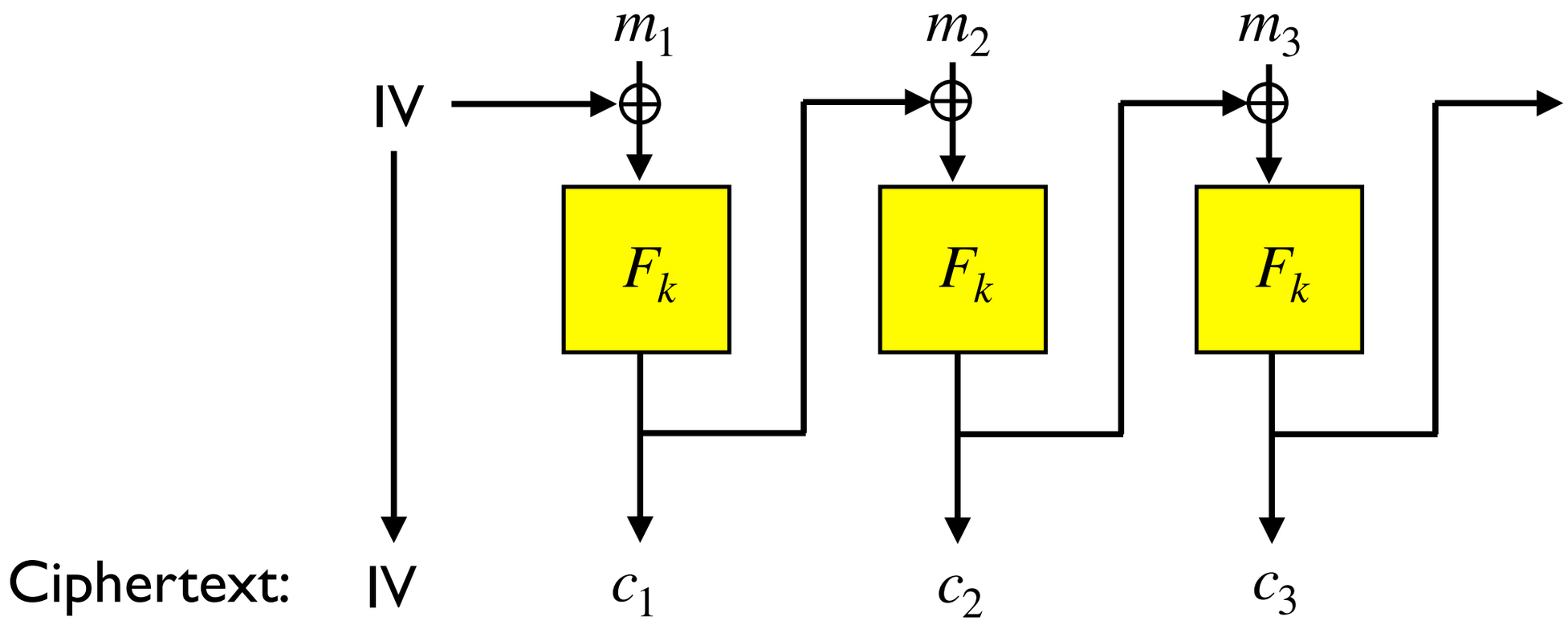
Vote: Which is more secure? (CTR/CBC/the same)

This class is being recorded

Block Cipher CBC Mode

Another popular option is known as CBC mode. It requires F_k which is a pseudorandom permutation. It is CPA-secure.

As with CTR mode, break the message up into blocks of size n and use a single IV.



To decrypt, Bob must invert F_k on block i and XOR with c_{i-1}.

Vote: Which is more secure? (CTR/CBC/the same)

This class is being recorded