CMSC/Math 456: Cryptography (Fall 2023)

Lecture 5

Daniel Gottesman

Administrative

Reminder: Problem Set 2 due on Thursday (Sep. 14) at noon, again on Gradescope.

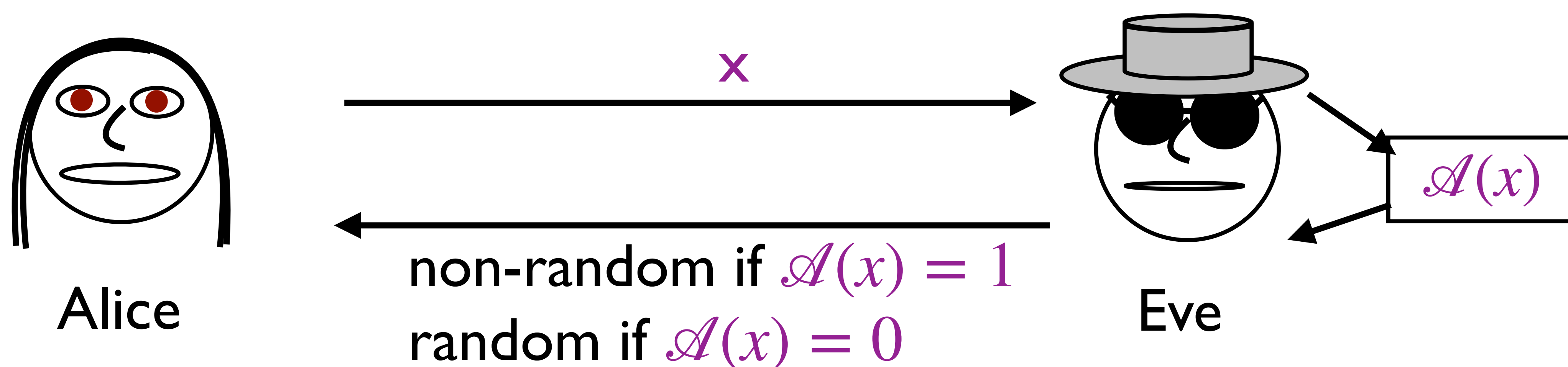
This class is being recorded

Pseudorandom Generators

Definition: Let $G : \{0,1\}^s \to \{0,1\}^{\ell(s)}$ be a *deterministic efficient* 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 $\{0,1\}^{\ell(s)}$.

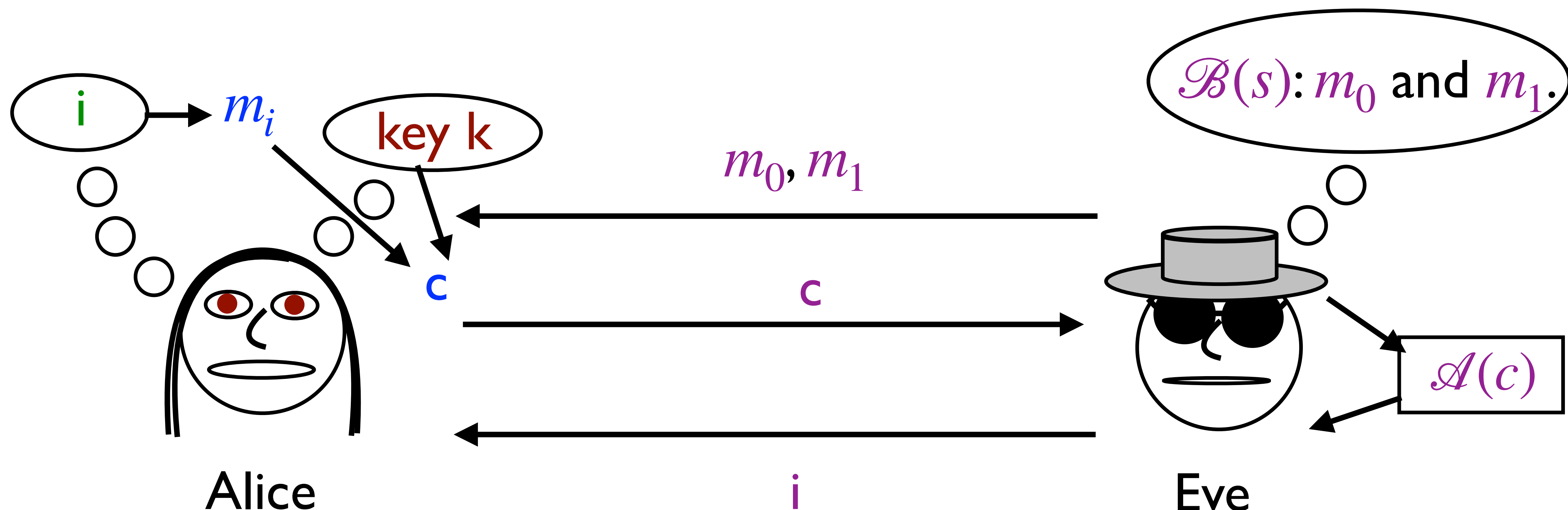


EAV Security

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.



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

$\mathscr{A}'(x)$ will succeed in distinguishing the pseudorandom numbers from random with probability $\epsilon'(s)$ if $(\mathscr{B}(s), \mathscr{A}(c))$ 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.

Basic Concept of a Reduction

At its heart, a reduction is a statement of the form "If you can do A, then you can do B."

Examples:

- If you can boil water, then you can cook an egg.
- If you can roller blade, then you can ice skate.
- If you can ice skate, then you can roller blade.
- If you can breathe water, then you can visit sunken ships.
- If you can go faster than light, you can travel back in time.

But this doesn't really work:

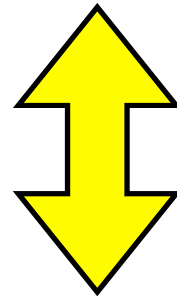
- If you can breathe water, then you can fly to the moon.

because it is non-constructive.

A reduction is supposed to tell you how doing A lets you do B.

Contrapositive of a Reduction

If you can do A, then you can do B.



If you can't do B, then you can't do A.

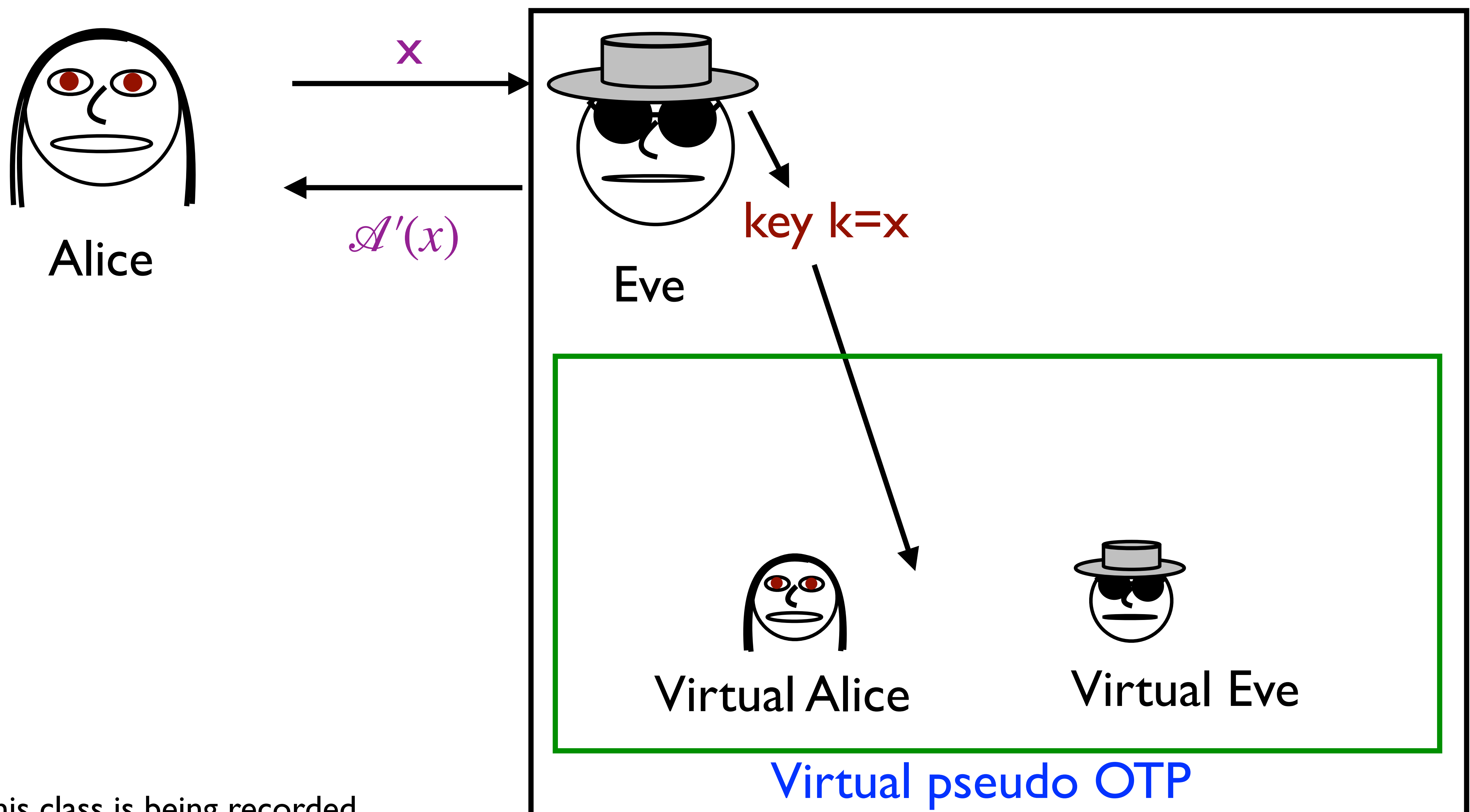
We most often prove the "A implies B" statement of a reduction but most often use the "not B implies not A" contrapositive.

E.g.:

- If you can't roller blade, then you must not be able to ice skate.
- If you can't break the pseudorandom generator, then you can't break the pseudo one-time pad.

The Reduction

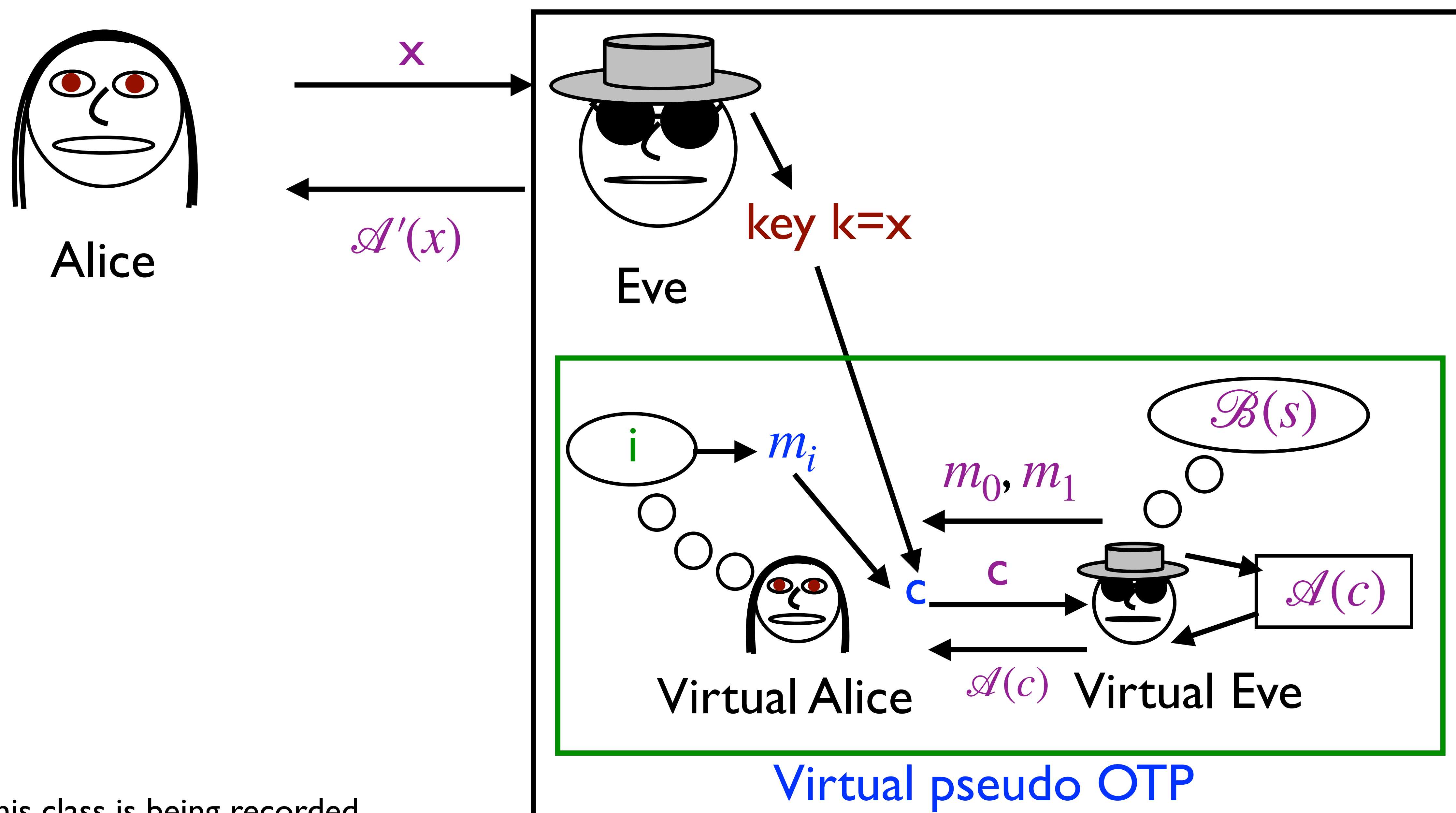
The idea is that the attack $\mathscr{A}'(x)$ will use x as the key in a virtual pseudo one-time pad protocol, in which a virtual Eve will run the attack $(\mathscr{B}(s),\mathscr{A}(c))$ on the pseudo one-time pad.



This class is being recorded

The Reduction

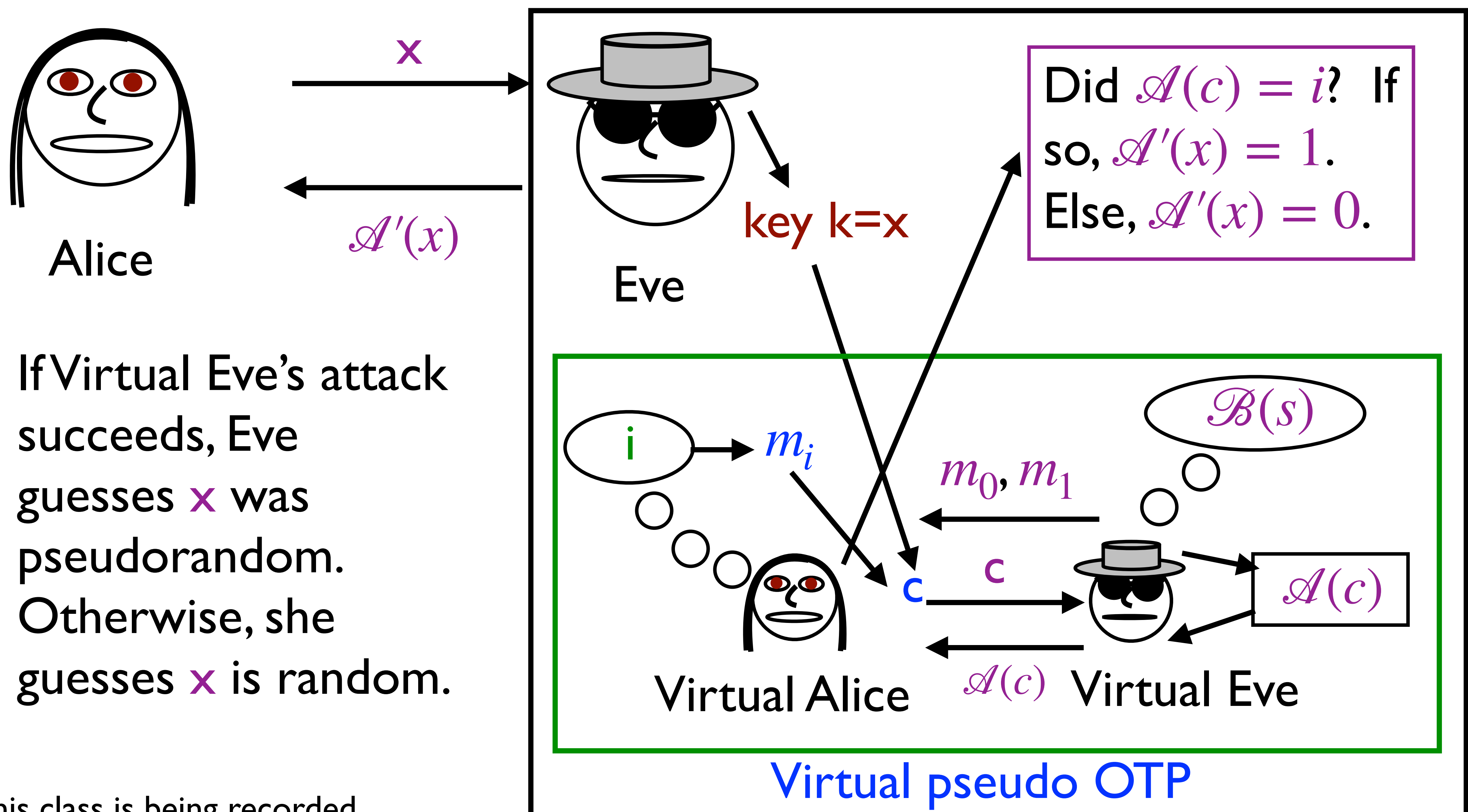
The idea is that the attack $\mathscr{A}'(x)$ will use x as the key in a virtual pseudo one-time pad protocol, in which a virtual Eve will run the attack $(\mathscr{B}(s),\mathscr{A}(c))$ on the pseudo one-time pad.



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.



Proof Steps

To prove the result, we will need analyze the reduction carefully.

1. First, we will give a formula for the chance that $\mathscr{A}'(k) = 1$ for key k.
2. Next, compute the probability that the attack $\mathscr{A}'(x)$ succeeds in the case where x was random.
3. Then we compute the probability that the attack $\mathscr{A}'(x)$ succeeds in the case where x was pseudorandom and compare with the case where x was random.
4. Finally, apply the definition of pseudorandomness to show that the two are close, which then implies that the attack has a low chance of success against the pseudo one-time pad.
5. We also analyze the reduction to show that the derived attack is efficient.

Probability Distribution for $\mathscr{A}'(k)$

Let calculate the probability for $\mathscr{A}'(k) = 1$ when the key passed to the simulation is k drawn from one of the two distributions $\mathscr{K}$ (random or pseudorandom):

$$\begin{aligned} \Pr(\mathscr{A}'(k) = 1) &= \frac{1}{2}\left[\Pr(\mathscr{A}(Enc(k, m_1)) = 1) + \Pr(\mathscr{A}(Enc(k, m_0)) = 0)\right] \\ &= \frac{1}{2}\left[\Pr(\mathscr{A}(Enc(k, m_1)) = 1) + 1 - \Pr(\mathscr{A}(Enc(k, m_0)) = 1)\right] \\ &= \frac{1}{2}\left(1 + \delta_{\mathscr{K}}\right) \end{aligned}$$

with probabilities averaged over $k \in \mathscr{K}$ and randomness in Enc and

$$\delta_{\mathscr{K}} = \Pr(\mathscr{A}(Enc(x, m_1)) = 1) - \Pr(\mathscr{A}(Enc(x, m_0)) = 1)$$

(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.)

Success Probability for Random x

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_0, m_1) was chosen by $\mathscr{B}(s)$ and whichever ciphertext c ended up being used,

$$\Pr(Enc(x, m_0) = c) = \Pr(Enc(x, m_1) = c) = p_c$$

This class is being recorded

Success Probability for Random x

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_0, m_1) was chosen by $\mathscr{B}(s)$ and whichever ciphertext c ended up being used,

$$\Pr(Enc(x, m_0) = c) = \Pr(Enc(x, m_1) = c) = p_c$$

Now,

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

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

which doesn't depend on i. In particular,

$$\delta_{\text{rand}} = \Pr(\mathscr{A}(Enc(x, m_1)) = 1) - \Pr(\mathscr{A}(Enc(x, m_0)) = 1) = 0$$

and $\Pr(\mathscr{A}'(x) = 1) = 1/2$.

This class is being recorded

Pseudorandom vs. Random

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

$$\delta_{\mathrm{PRG}}(y) = \Pr(\mathscr{A}(Enc(G(y), m_1)) = 1) - \Pr(\mathscr{A}(Enc(G(y), m_0)) = 1)$$

$$\Pr(\mathscr{A}'(G(y)) = 1) = 1/2\,(1 + \delta_{\mathrm{PRG}})$$

Pseudorandom vs. Random

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

$$\delta_{\mathrm{PRG}}(y) = \Pr(\mathscr{A}(Enc(G(y), m_1)) = 1) - \Pr(\mathscr{A}(Enc(G(y), m_0)) = 1)$$

$$\Pr(\mathscr{A}'(G(y)) = 1) = 1/2\,(1 + \delta_{\mathrm{PRG}})$$

Now compare the random and pseudorandom cases:

$$\begin{aligned}\Pr_y(\mathscr{A}'(G(y)) = 1) - \Pr_x(\mathscr{A}'(x) = 1) &= \frac{1}{2}(1 + \delta_{\mathrm{PRG}}) - \frac{1}{2} \\ &= \frac{1}{2}\delta_{\mathrm{PRG}}\end{aligned}$$

Bound on Attack Success

The security definition for the pseudorandom generator says

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

for negligible $\epsilon(s)$. Thus,

$$|\delta_{\mathrm{PRG}}| \leq 2\epsilon(s)$$

Bound on Attack Success

The security definition for the pseudorandom generator says

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

for negligible $\epsilon(s)$. Thus,

$$|\delta_{\text{PRG}}| \leq 2\epsilon(s)$$

or, expanding δ_{PRG} and letting $k = G(y)$,

$$|\Pr_k(\mathscr{A}(Enc(k, m_1)) = 1) - \Pr_k(\mathscr{A}(Enc(k, m_0)) = 1)| \leq 2\epsilon(s)$$

This is precisely the definition of EAV security for the pseudo one-time pad since $2\epsilon(s)$ is negligible if $\epsilon(s)$ is.

Complexity of Reduction

One additional thing that needs to be checked is the complexity of the attack $\mathscr{A}'(x)$.

- Running the Virtual Alice in the one-time pad simulation takes a time linear in the message length $O(\ell(s))$.
- Running Virtual Eve takes the time required to run the attack $(\mathscr{B}(s), \mathscr{A}(c))$, which is polynomial in the message length.
- Translating the outcome of the simulated one-time pad into the attack $\mathscr{A}'(x)$ takes constant time.

Therefore the whole attack takes time polynomial in $\ell(s)$, which is as it should be.

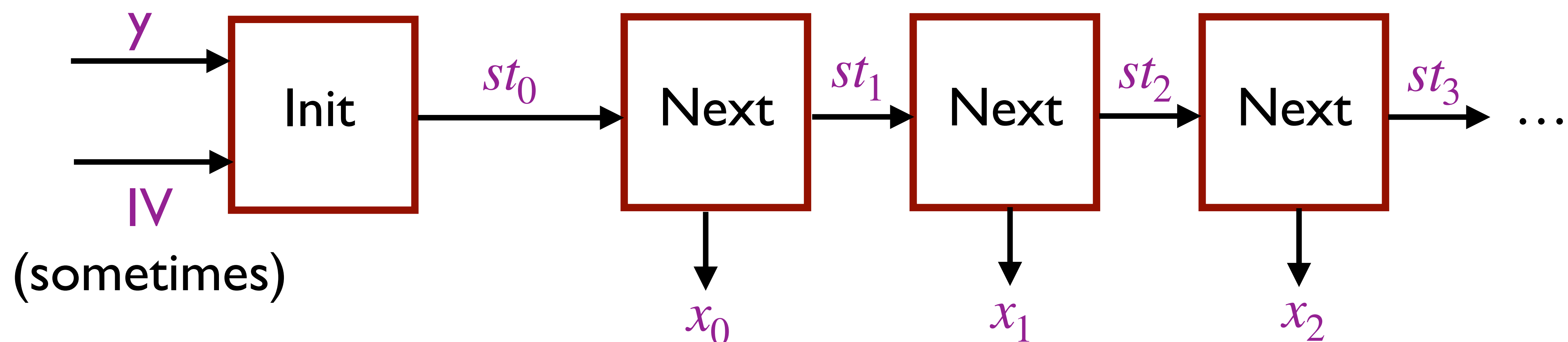
Note: "Polynomial time" mean polynomial in the size of the input to the algorithm, which is here $\ell(s)$, not necessarily polynomial in s.

This class is being recorded

Stream Ciphers

Pseudorandom generators are a lot more efficient than using truly random bits, but are still a bit clunky in that they can only output big chunks of bits. You need to know how many bits $\ell(s)$ you are going to need when you pick s (and therefore when you establish your key if you are doing the pseudo one-time pad.

Stream ciphers are a solution: given a seed, they generate a stream of pseudorandom bits as you desire.



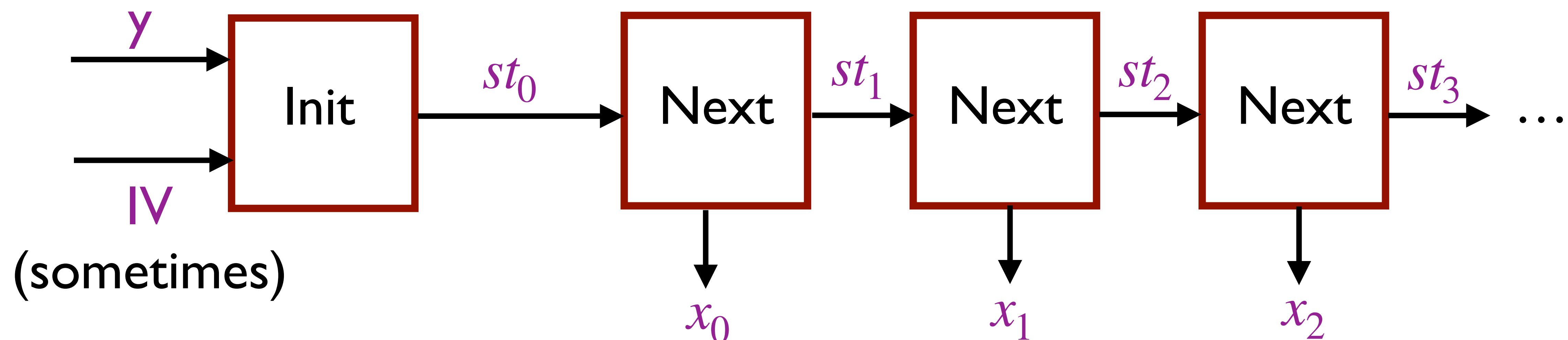
This class is being recorded

Stream Cipher Definition

A stream cipher is a pair of *deterministic efficient* algorithms (Init, Next).

- Init is the initialization algorithm; it takes a seed y as input and sometimes an initial value IV and outputs an initial state st_0.
- Next is the algorithm that produces bits; it takes as input a current state st_i and outputs a bit x_i and a new state st_{i+1}.

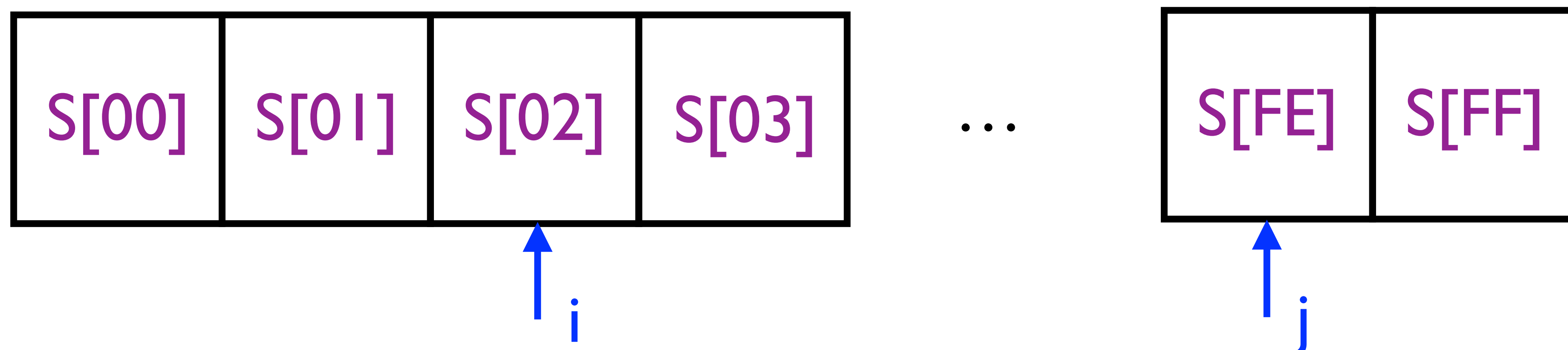
When there is no IV, the stream cipher is secure if the function $G_n(y) = x_0 x_1 x_2 \ldots x_n$ is a pseudorandom generator for any n which is polynomial in $|y| = s$.



This class is being recorded

RC4 Overview

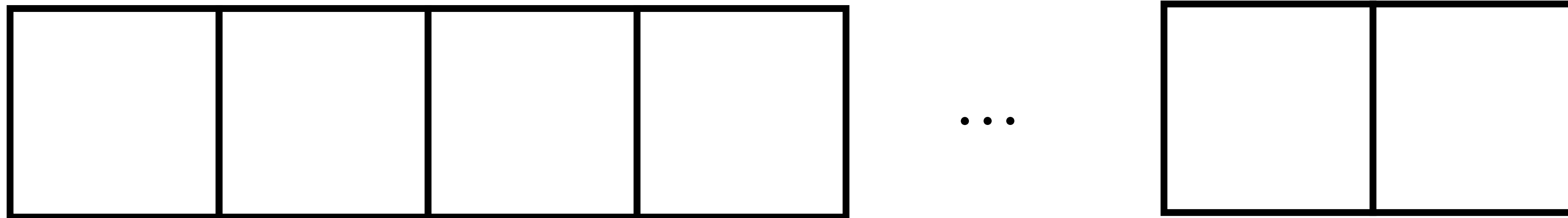
Invented by Ron Rivest, RC4 was widely used for many years, for instance as part of the first wi-fi protocol, WEP.



RC4 is a stream cipher. The state passed between steps consists of a permutation of the numbers 0…255 (realized as an array of 256 bytes, each containing a distinct number), plus two additional bytes i and j, which are pointers into the array.

It works by performing swaps between locations in the array in a somewhat complicated way. One entry is then returned as the next output in the stream.

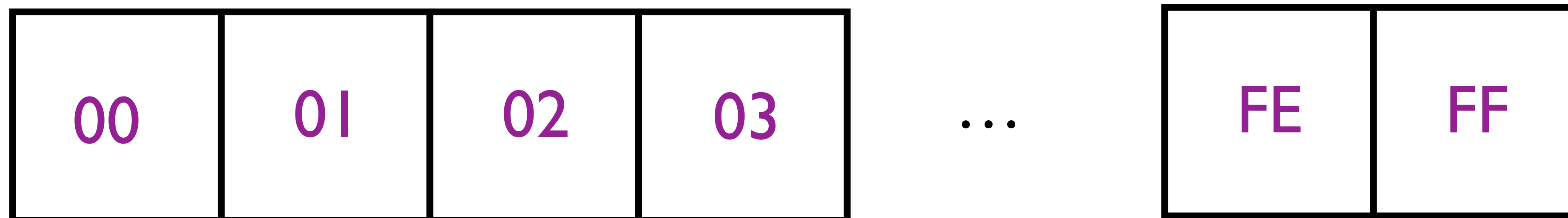
RC4 Init



$k = (03, FA, \ldots)$

RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the ($i \bmod s$)th byte of k when $i \geq s$.

RC4 Init

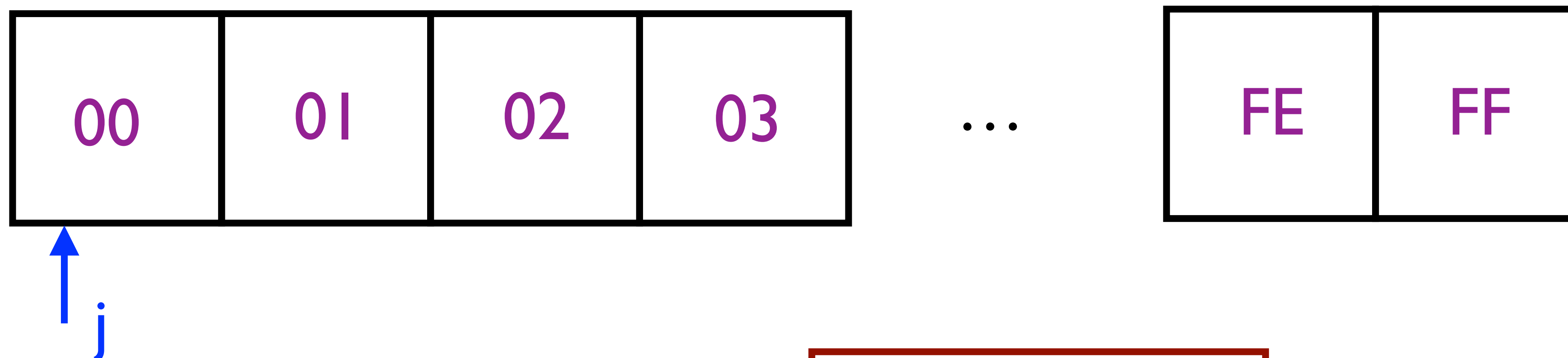


$k = (03, FA, \ldots)$

RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the $(i \bmod s)$th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.

RC4 Init



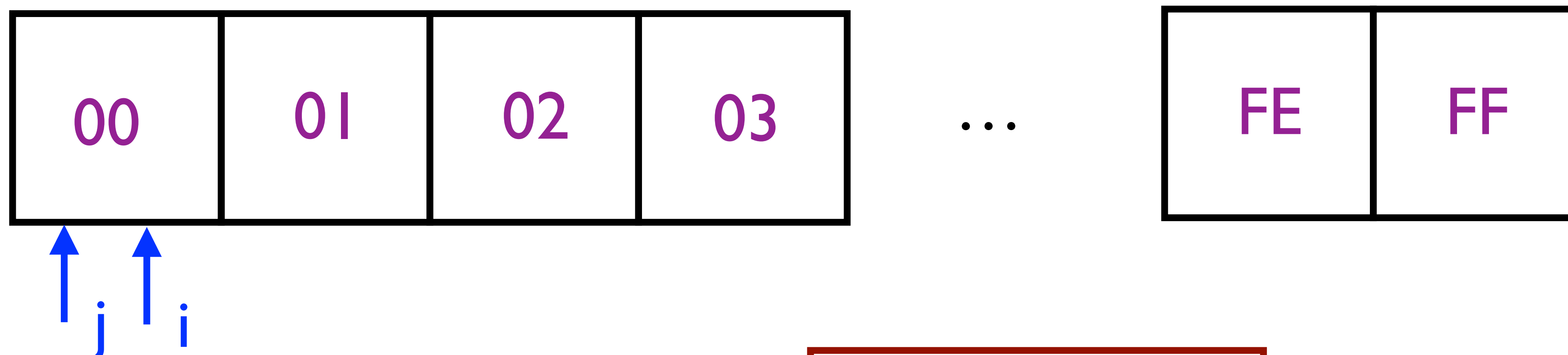
$k = (03, FA, \ldots)$

RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the $(i \bmod s)$th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.

This class is being recorded

RC4 Init



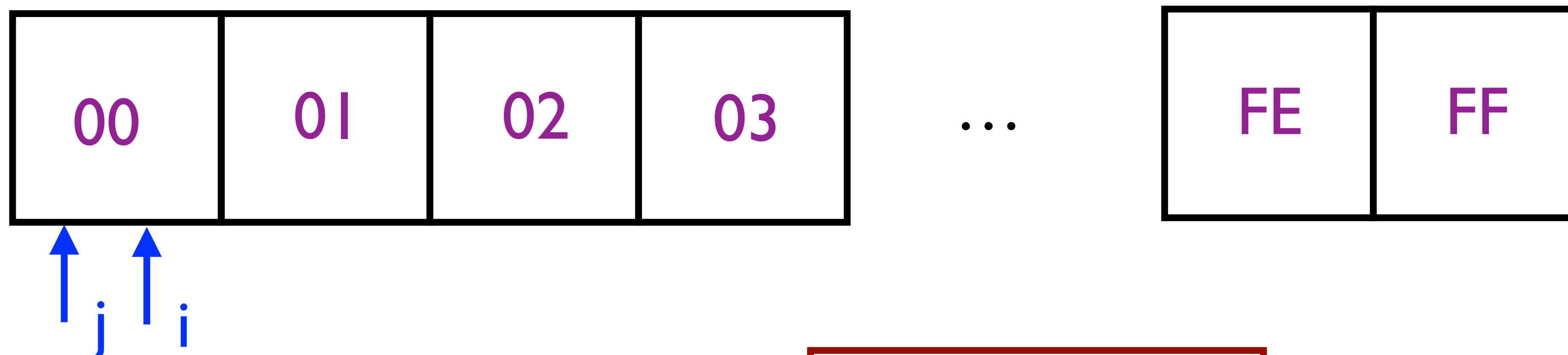
$k = (03, FA, \ldots)$

RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the $(i \bmod s)$th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:

This class is being recorded

RC4 Init

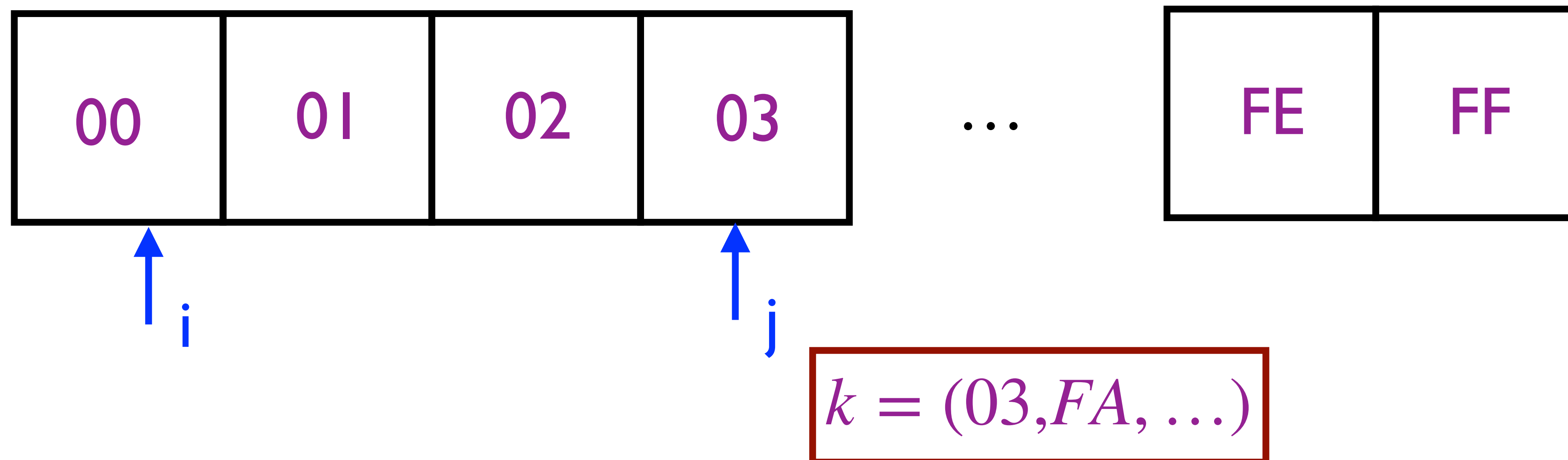


$k = (03, FA, \ldots)$

RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the (i mod s)th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + k_i mod 256

RC4 Init

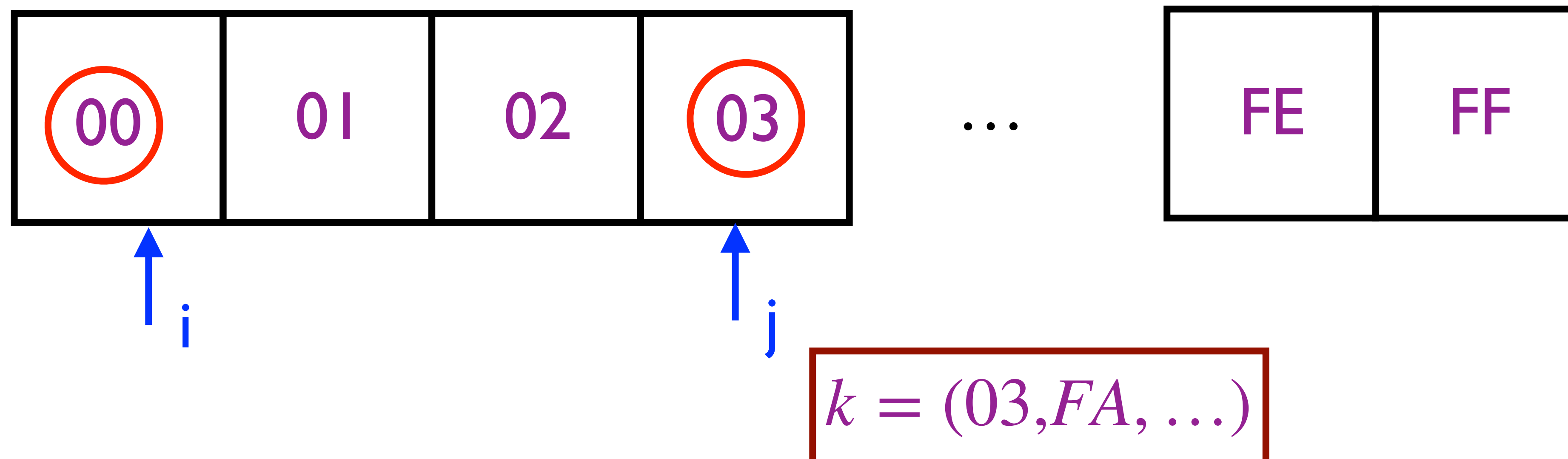


RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the ($i \bmod s$)th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + $k_i \bmod 256$

This class is being recorded

RC4 Init

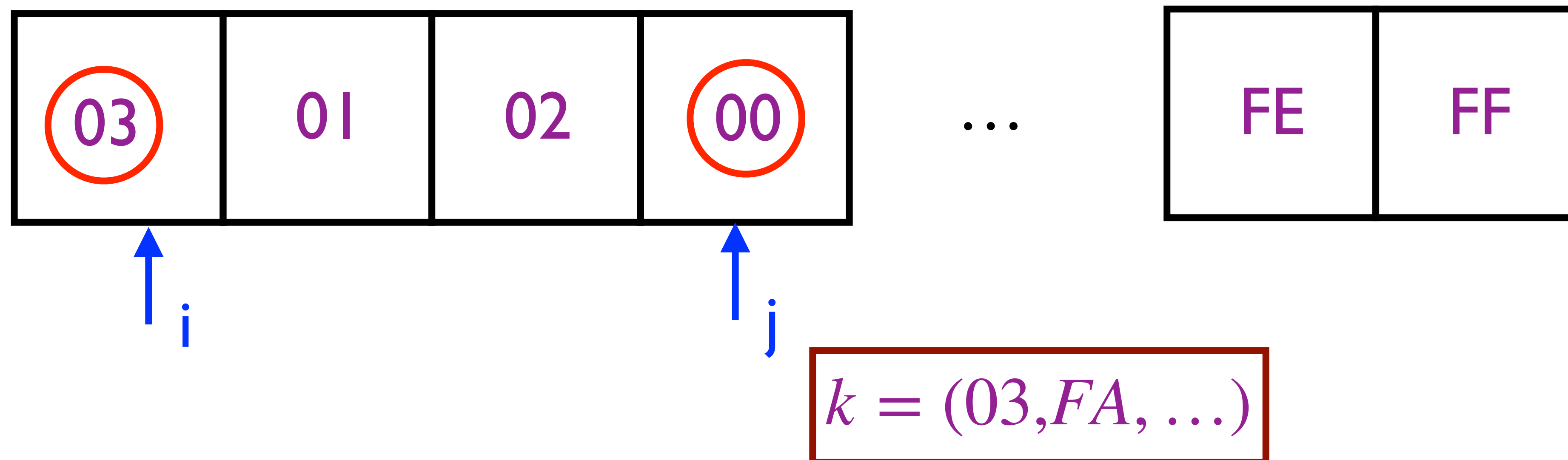


RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the ($i \bmod s$)th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + $k_i \bmod 256$
  - Swap S[i] and S[j].

This class is being recorded

RC4 Init

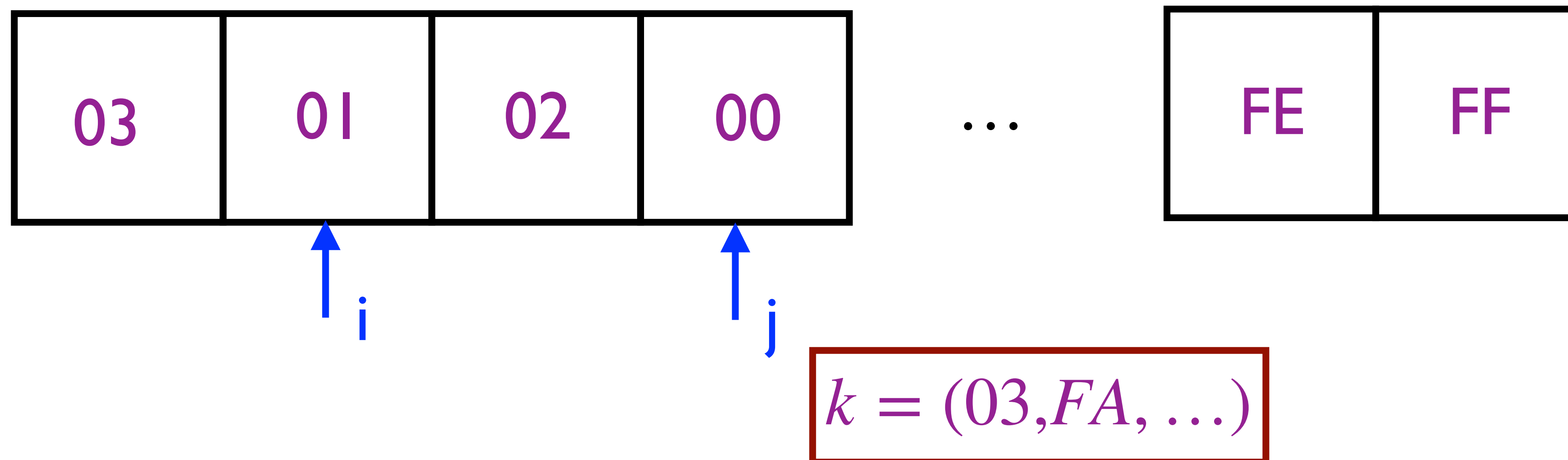


RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the ($i \bmod s$)th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + $k_i \bmod 256$
  - Swap S[i] and S[j].

This class is being recorded

RC4 Init

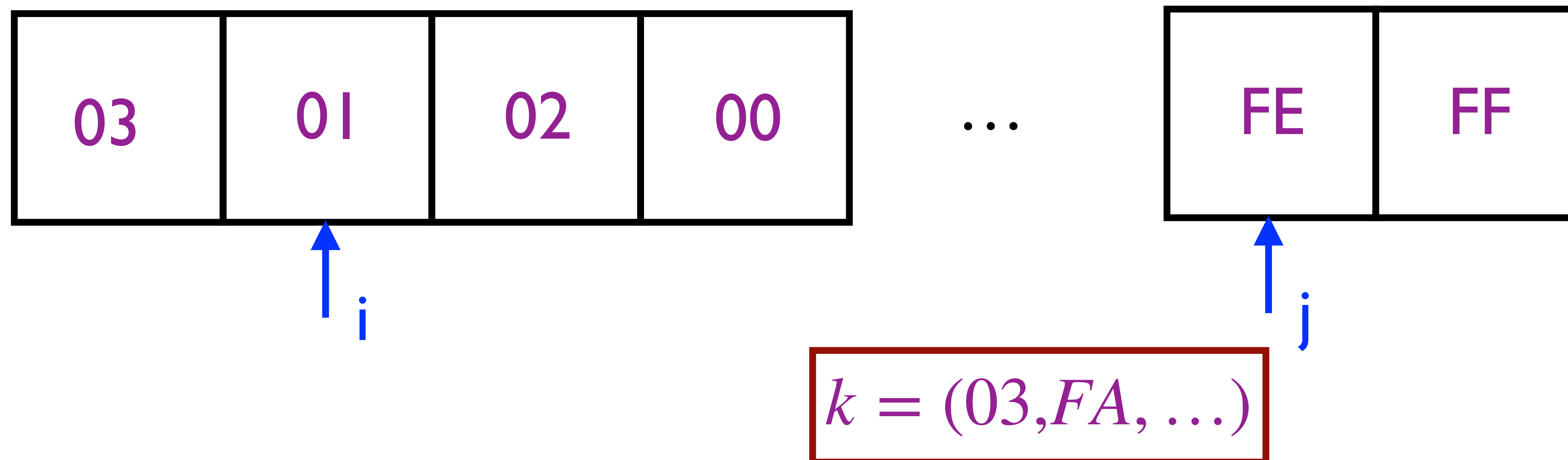


$k = (03, FA, \ldots)$

RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the ($i \bmod s$)th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + $k_i \bmod 256$
  - Swap S[i] and S[j].

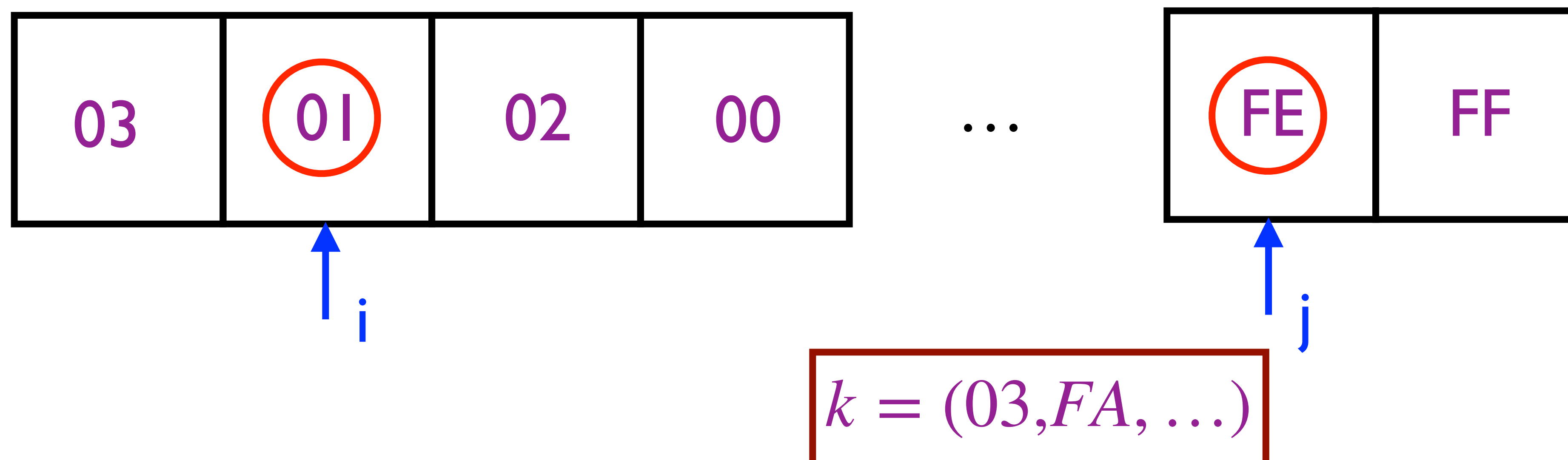
RC4 Init



RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the ($i \bmod s$)th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + $k_i \bmod 256$
  - Swap S[i] and S[j].

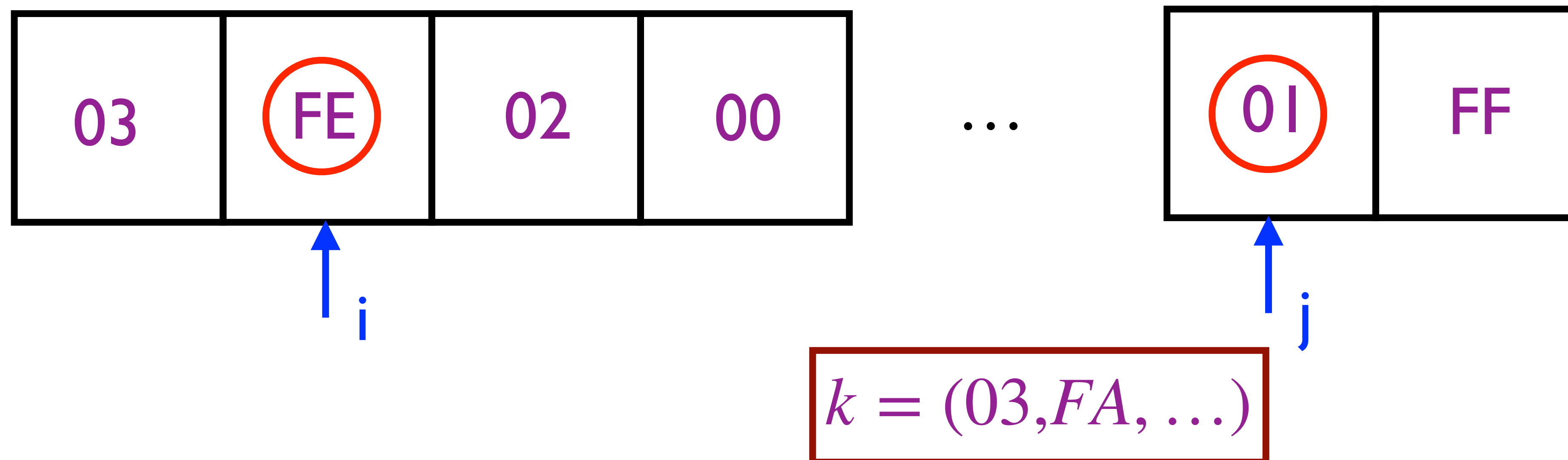
RC4 Init



RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the $(i \bmod s)$th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + k_i mod 256
  - Swap S[i] and S[j].

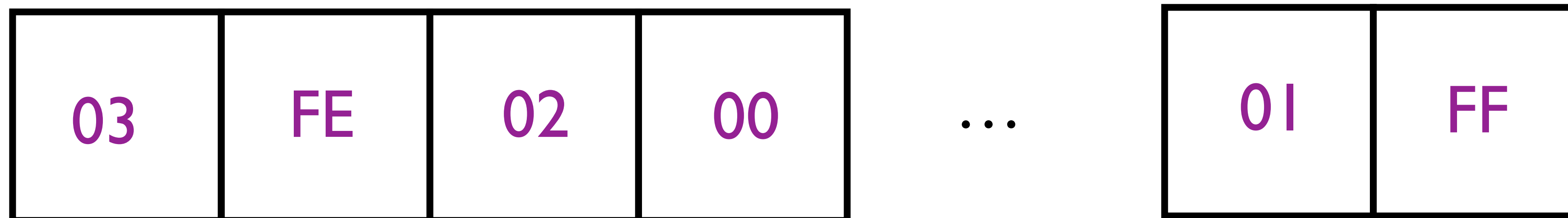
RC4 Init



RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the ($i \bmod s$)th byte of k when $i \geq s$.

- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + $k_i \bmod 256$
  - Swap S[i] and S[j].

RC4 Init

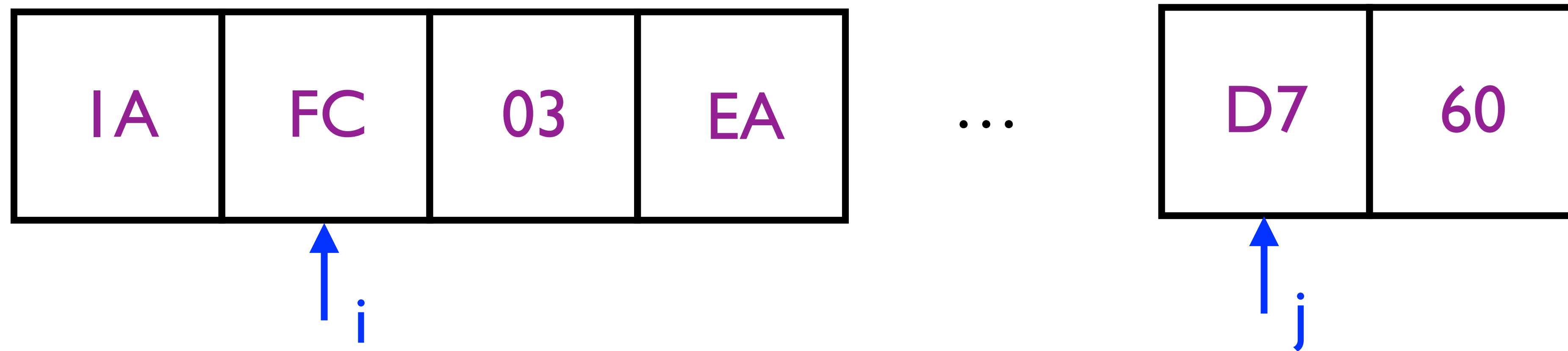


$k = (03, FA, \ldots)$

RC4 takes no IV, and the key k can have a variable length s up to 255 bytes. Let k_i be the ith byte of k if $i < s$, or the $(i \bmod s)$th byte of k when $i \geq s$.

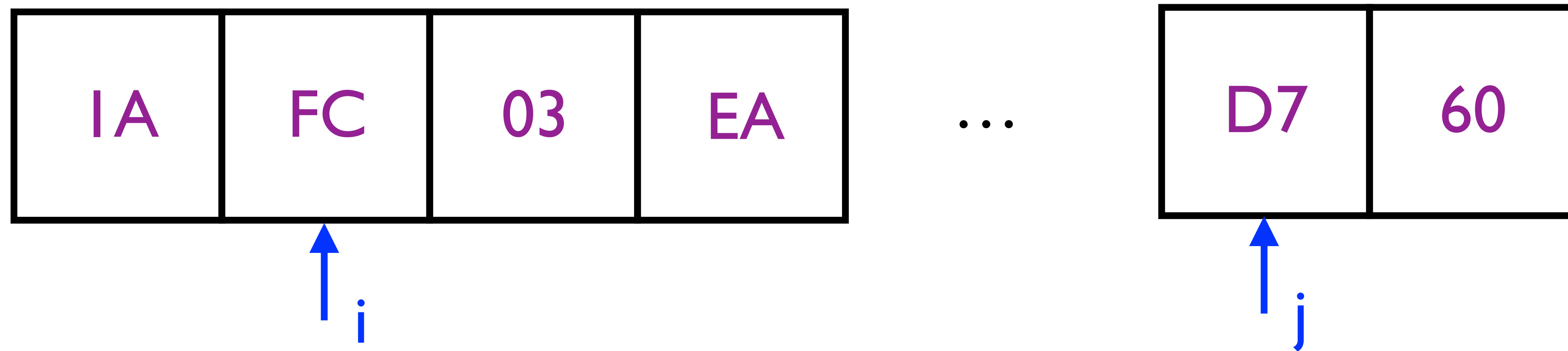
- First, initialize S[i] := i for all a=0, …, 255.
- Start with j := 0.
- For i = 0 to 255:
  - j := j + S[i] + $k_i \bmod 256$
  - Swap S[i] and S[j].
- Reset i := 0, j := 0.

RC4 Next



The Next algorithm has the following steps:

RC4 Next

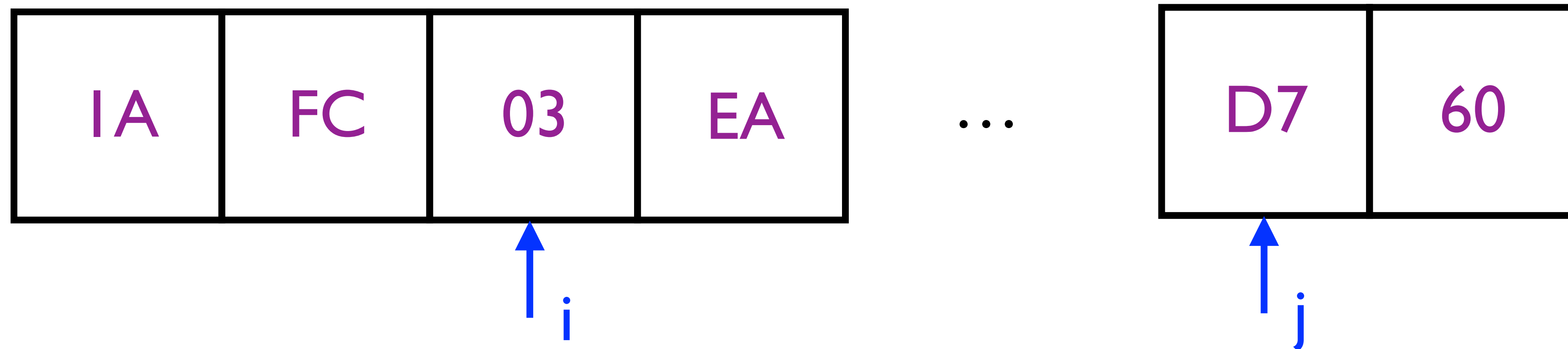


The Next algorithm has the following steps:

- i := i + 1

This class is being recorded

RC4 Next

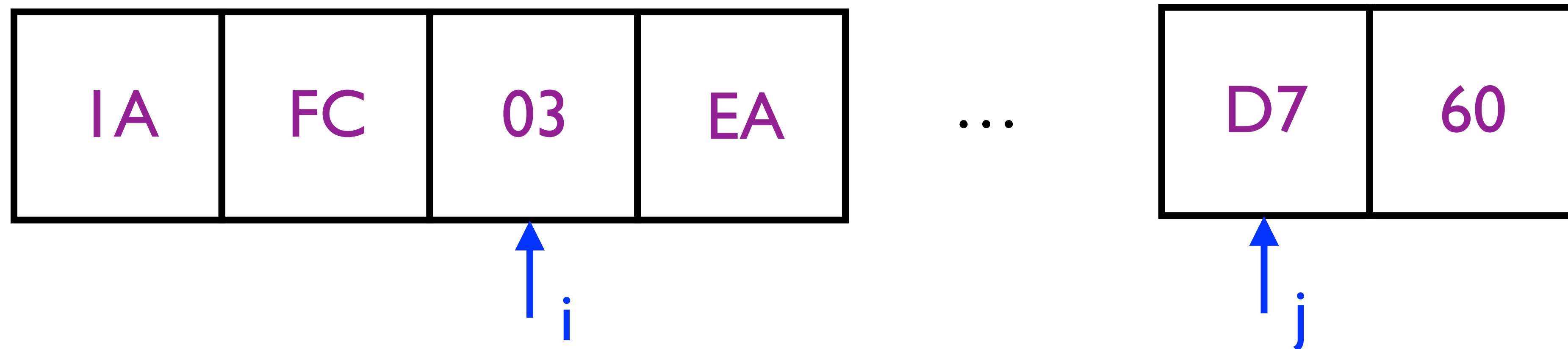


The Next algorithm has the following steps:

- i := i + 1

This class is being recorded

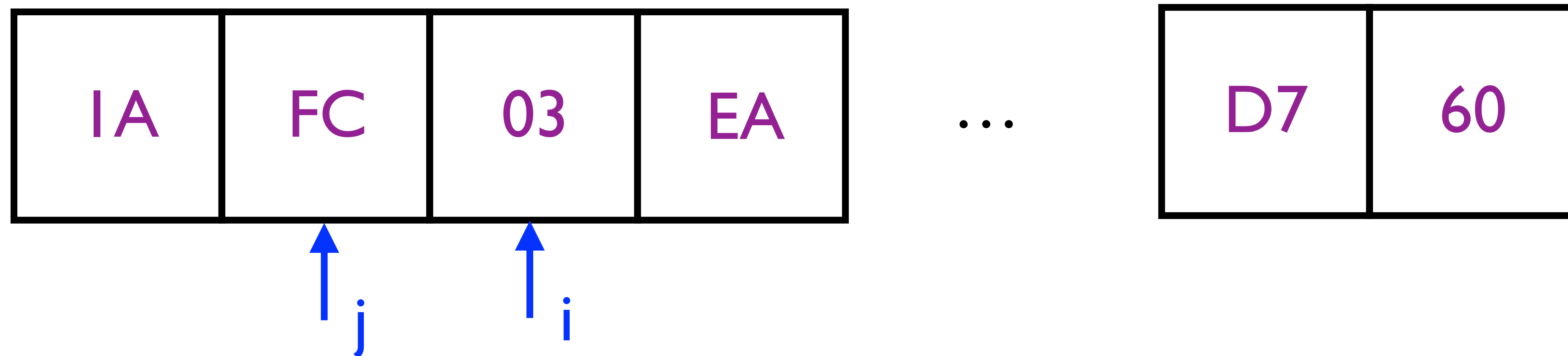
RC4 Next



The Next algorithm has the following steps:

- i := i + 1
- j := j + S[i]

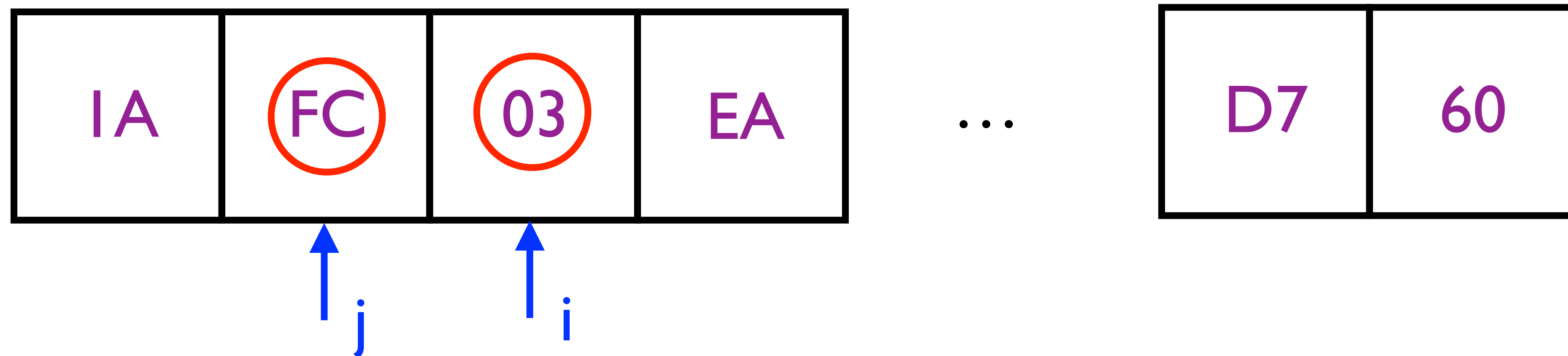
RC4 Next



The Next algorithm has the following steps:

- i := i + 1
- j := j + S[i]

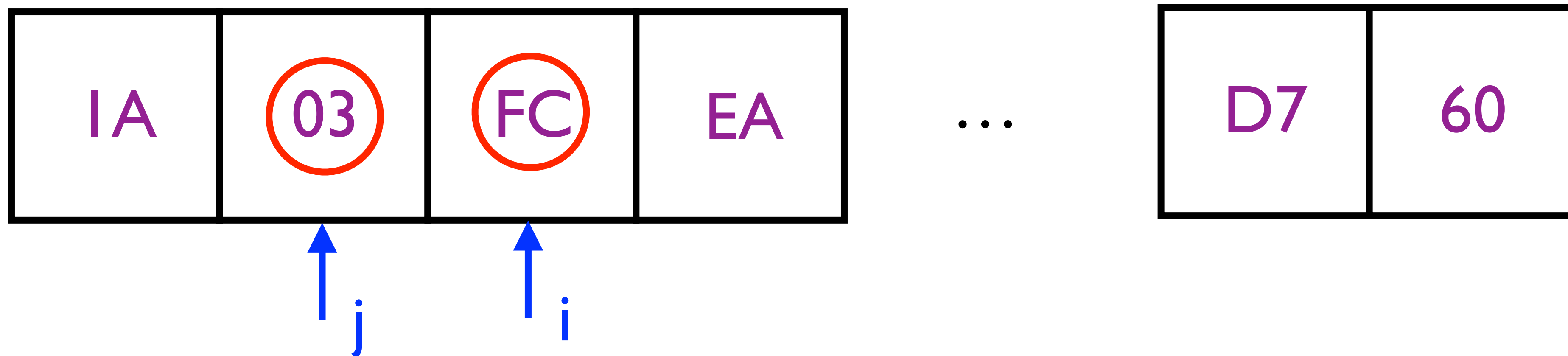
RC4 Next



The Next algorithm has the following steps:

- i := i + 1
- j := j + S[i]
- Swap S[i] and S[j]

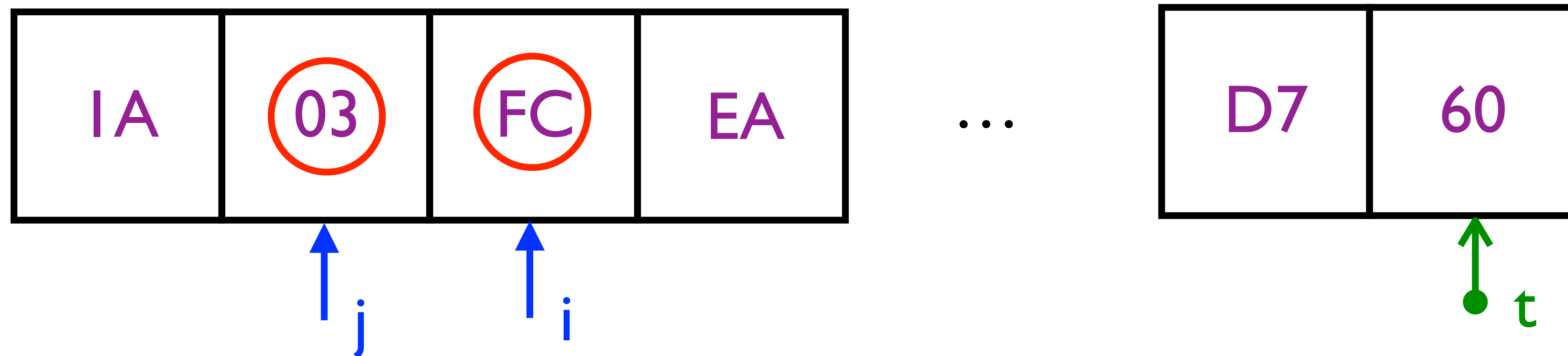
RC4 Next



The Next algorithm has the following steps:

- i := i + 1
- j := j + S[i]
- Swap S[i] and S[j]

RC4 Next



The Next algorithm has the following steps:

- i := i + 1
- j := j + S[i]
- Swap S[i] and S[j]
- t := S[i] + S[j]

RC4 Next

The Next algorithm has the following steps:

- i := i + 1
- j := j + S[i]
- Swap S[i] and S[j]
- t := S[i] + S[j]
- y := S[t]

This class is being recorded

RC4 Next

The Next algorithm has the following steps:

- i := i + 1
- j := j + S[i]
- Swap S[i] and S[j]
- t := S[i] + S[j]
- y := S[t]

Then y is the output byte and the state is passed to the following Next call.

The idea here is that if the entries S[i] are essentially random, then t is random too and y is the output of a location with no clear relation to i and j.

This class is being recorded

RC4 Security

Because it relies only on swaps and simple arithmetic, RC4 has simple fast implementations. But is it secure?

Vote: Can we prove RC4 is secure? (Yes/No)

This class is being recorded

RC4 Security

Because it relies only on swaps and simple arithmetic, RC4 has simple fast implementations. But is it secure?

Vote: Can we prove RC4 is secure? (Yes/No)

No, it would be hard to prove security because of how complicated it is, and we wouldn't in any case be able to prove security more than conditionally. But more importantly, this is a fixed-size protocol. The limit on key size is 256 bytes, so the time to attack it is a constant, at most about 256^{256}.

Instead security here means that, in practice, there is no known attack that is much better than brute force.

RC4 Security

Because it relies only on swaps and simple arithmetic, RC4 has simple fast implementations. But is it secure?

Vote: Can we prove RC4 is secure? (Yes/No)

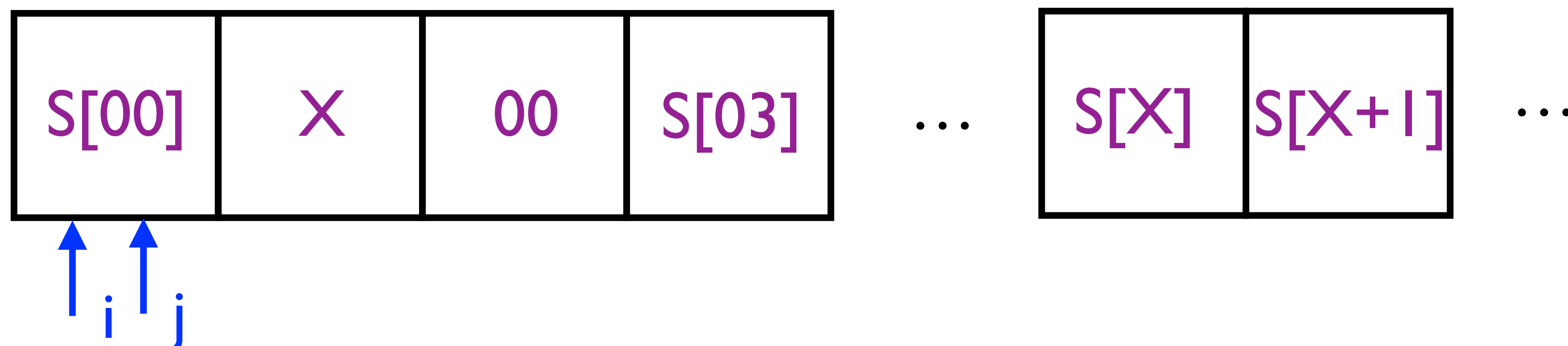
No, it would be hard to prove security because of how complicated it is, and we wouldn't in any case be able to prove security more than conditionally. But more importantly, this is a fixed-size protocol. The limit on key size is 256 bytes, so the time to attack it is a constant, at most about 256^{256}.

Instead security here means that, in practice, there is no known attack that is much better than brute force.

Vote: Is RC4 secure in this sense? (Yes/No)

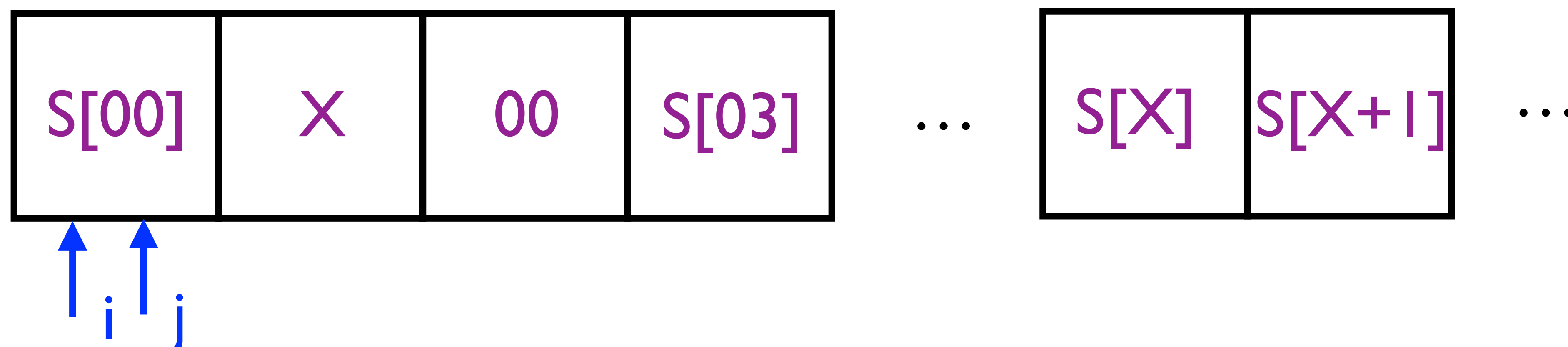
This class is being recorded

An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X ≠ 02. In this case,

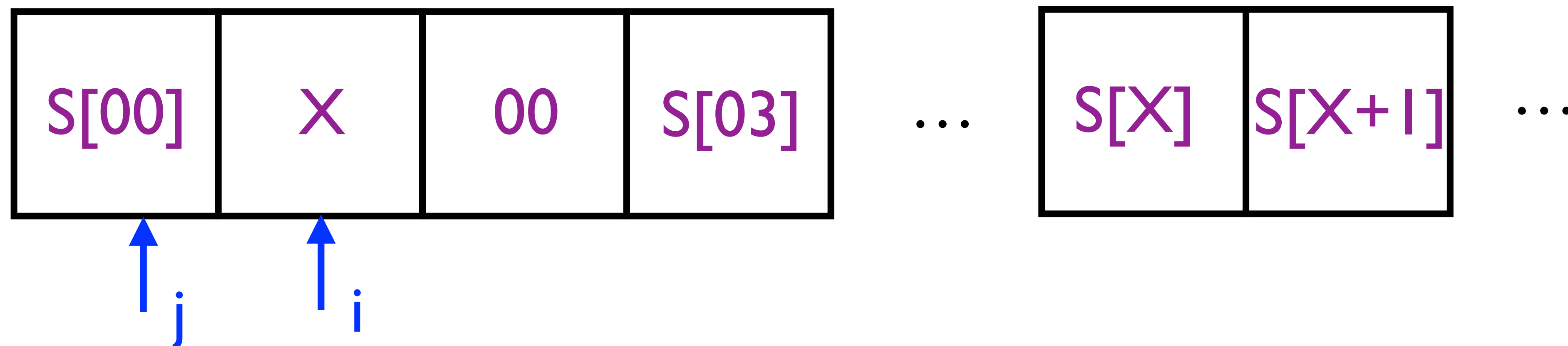
An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X ≠ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].

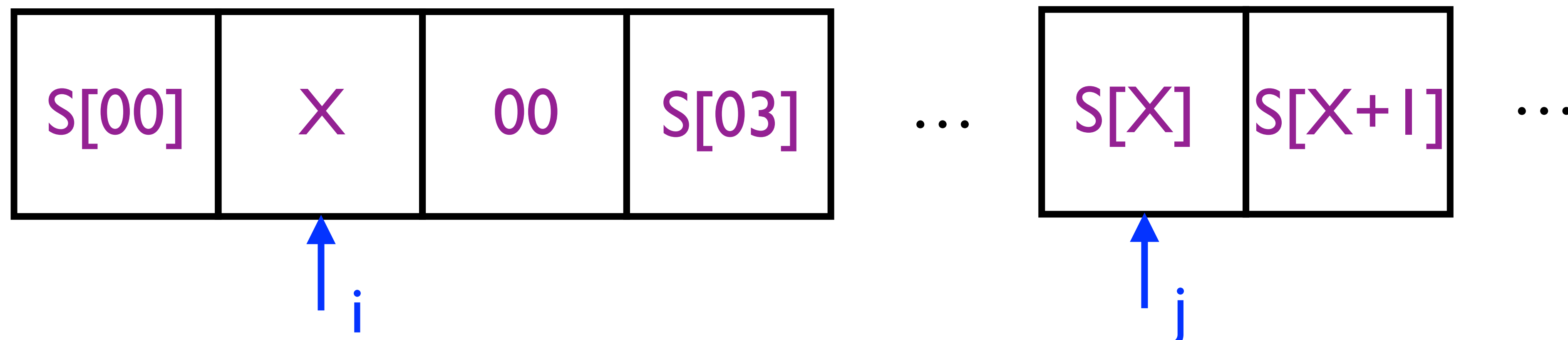
An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X ≠ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].

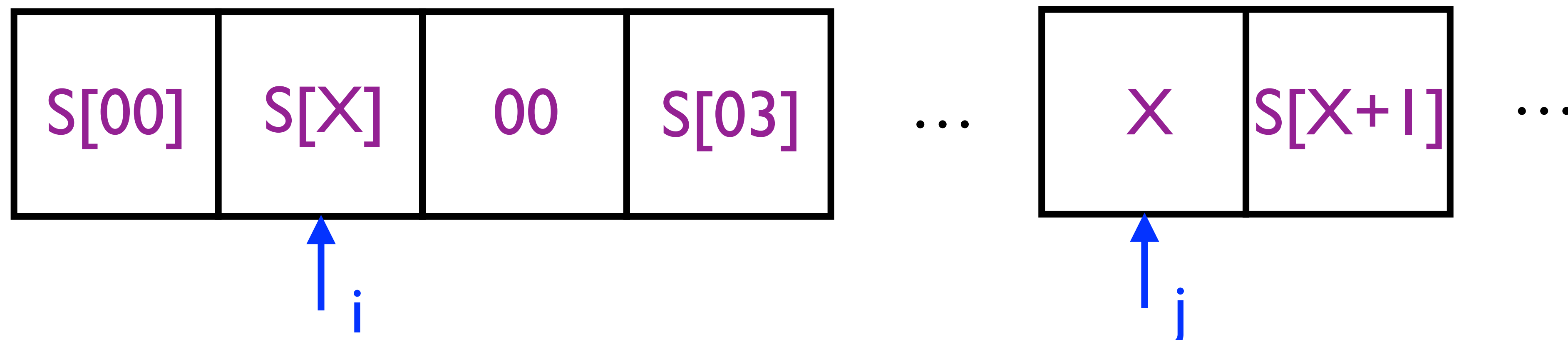
An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X $\neq$ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].

An Attack on RC4

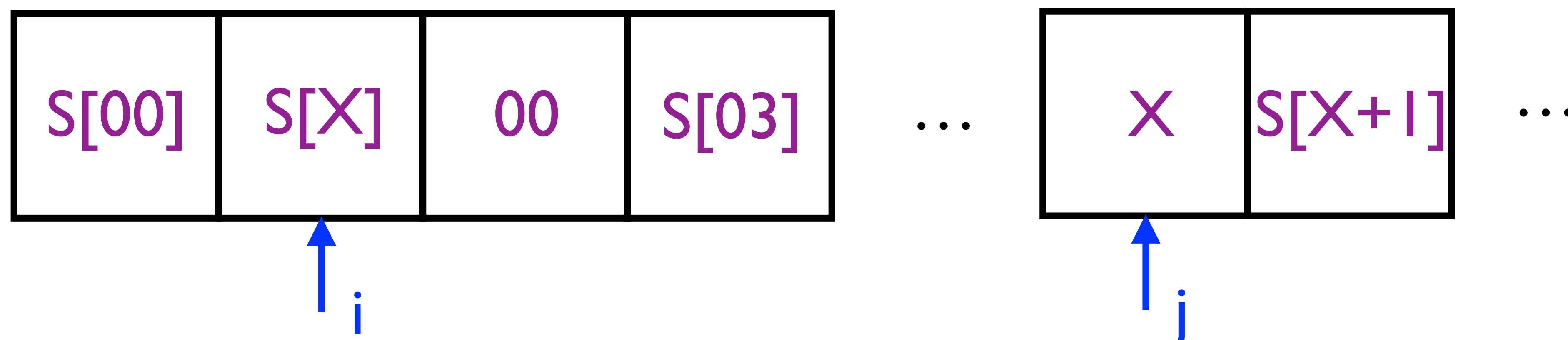


Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X ≠ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].

This class is being recorded

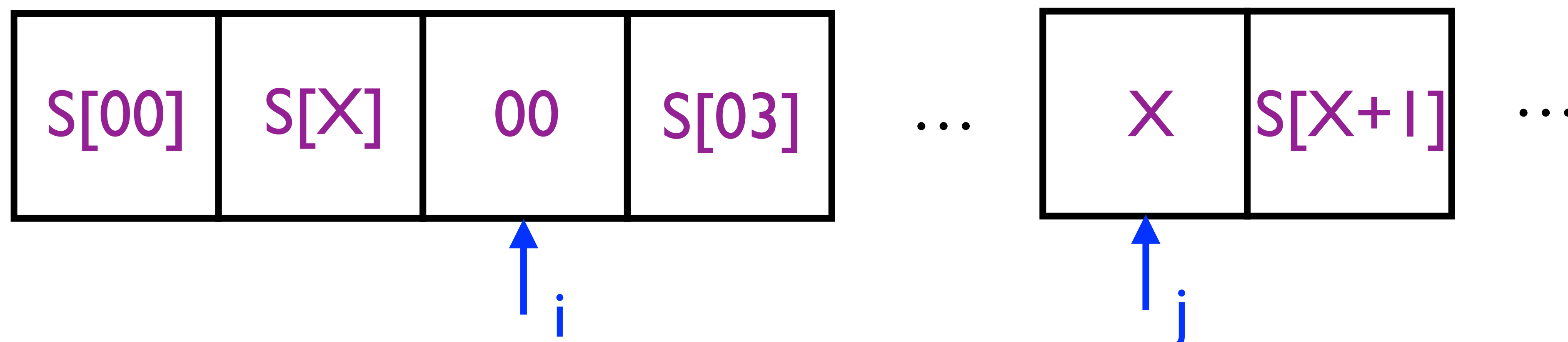
An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X ≠ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].
- The 2nd step sends i to 02, leaves j as X, and swaps S[02] and S[X]. Then t := S[i] + S[j] = X and y := S[t] = S[X] = 00.

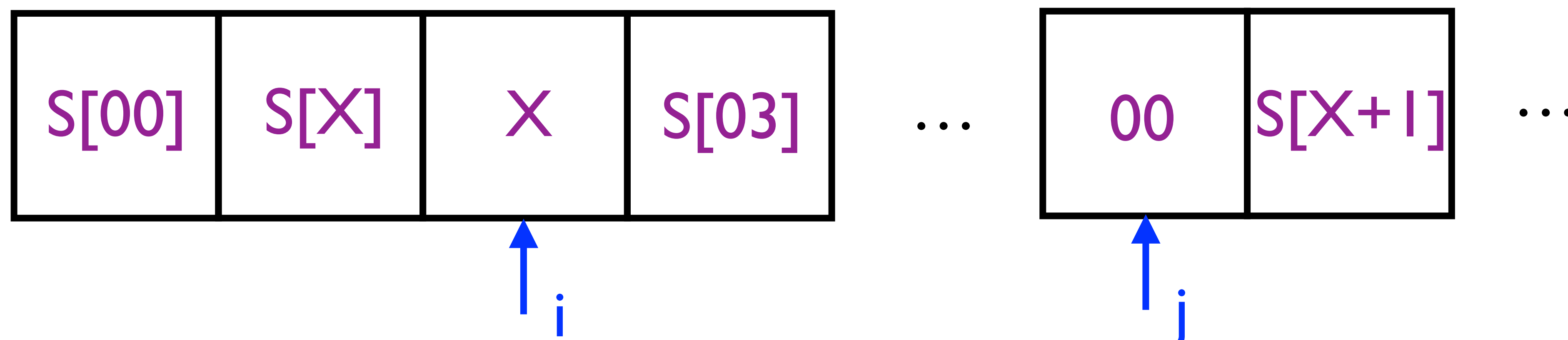
An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X ≠ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].
- The 2nd step sends i to 02, leaves j as X, and swaps S[02] and S[X]. Then t := S[i] + S[j] = X and y := S[t] = S[X] = 00.

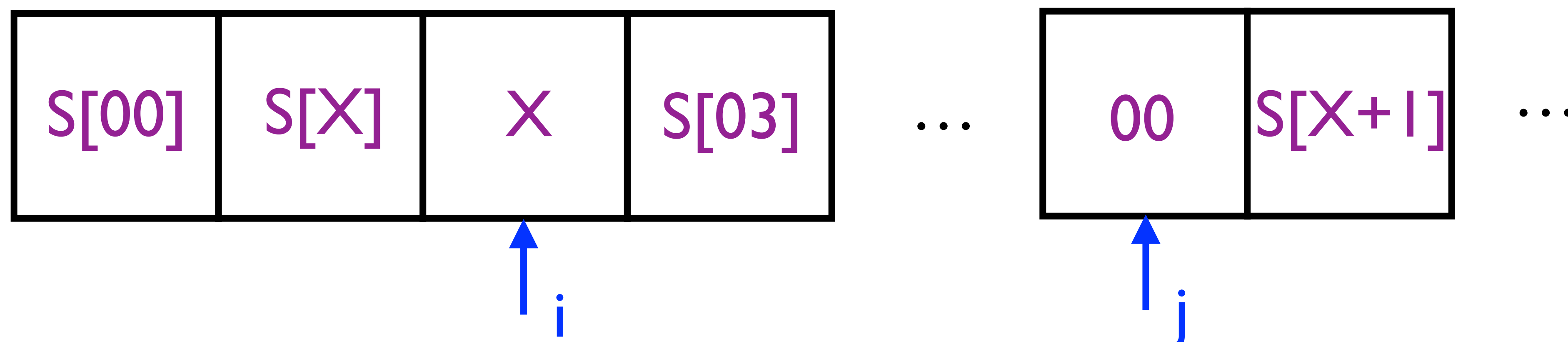
An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X $\neq$ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].
- The 2nd step sends i to 02, leaves j as X, and swaps S[02] and S[X]. Then t := S[i] + S[j] = X and y := S[t] = S[X] = 00.

An Attack on RC4



Suppose that after the Init algorithm, we can imagine the state of the array to be essentially a random permutation. With probability about 1/256, S[02] = 00 and S[01] = X ≠ 02. In this case,

- The 1st step sends i to 01, j to X, and swaps S[01] and S[X].
- The 2nd step sends i to 02, leaves j as X, and swaps S[02] and S[X]. Then t := S[i] + S[j] = X and y := S[t] = S[X] = 00.

If S[02] ≠ 00, t is essentially random, which leads to a 1/256 chance that y = 00. The output 00 is twice as likely as random!

This class is being recorded