Ciphers Over "Strange" Domains

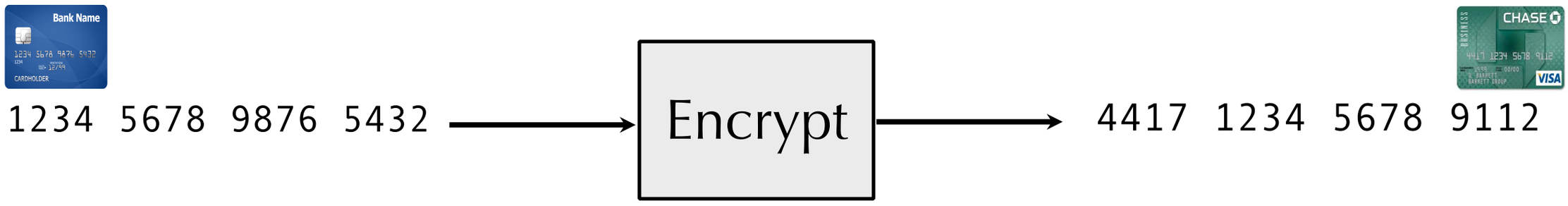
Tom Shrimpton

Florida Institute for Cybersecurity Research
University of Florida

UF Herbert Wertheim College of Engineering
Department of Computer & Information Science & Engineering
UNIVERSITY *of* FLORIDA

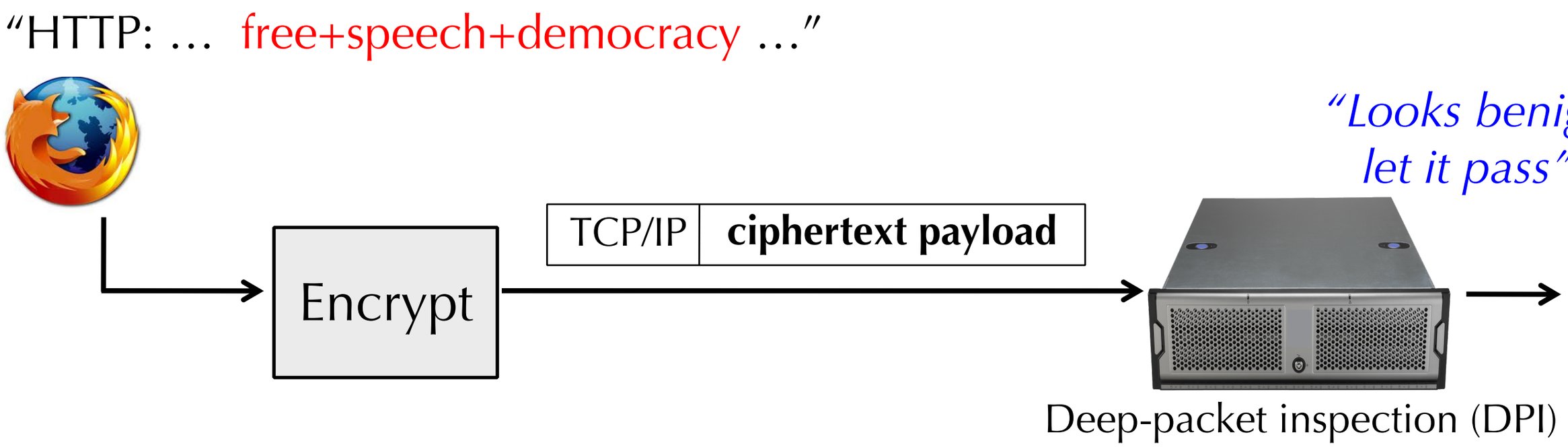
Today

In-place encryption of CC database



Thursday

Circumvention of nation-state internet censorship



Blockcipher basics

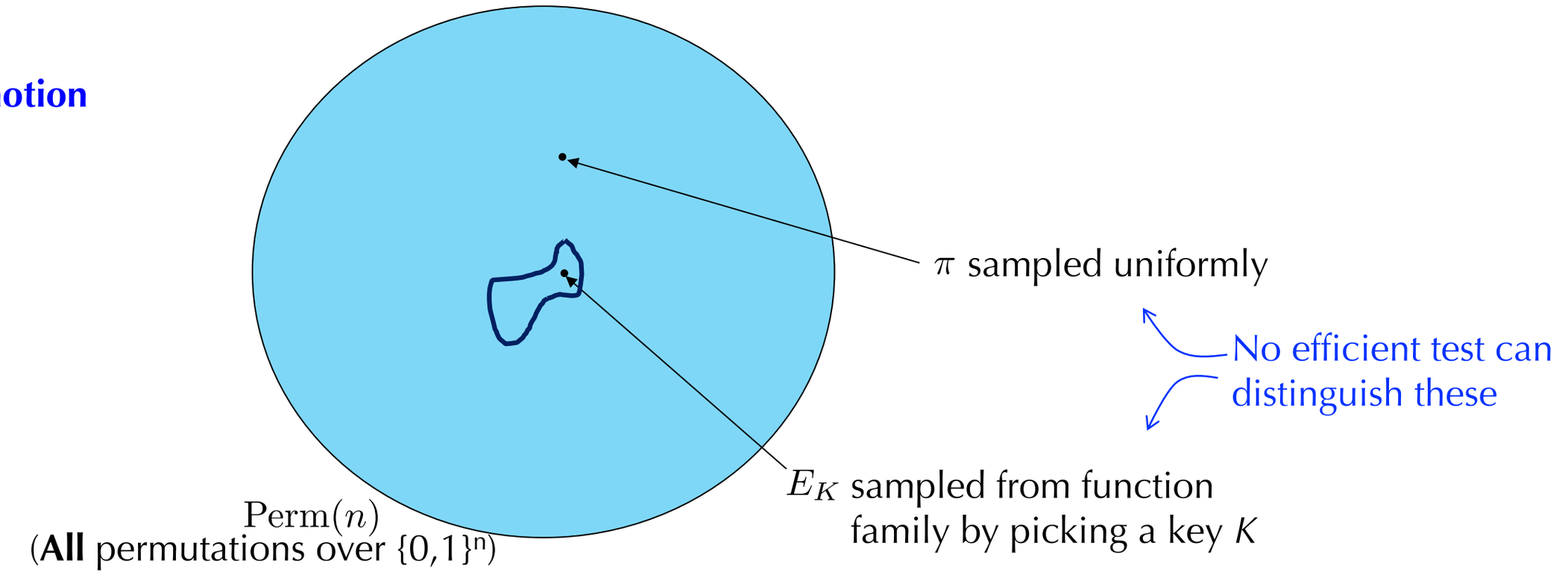
Traditional blockcipher syntax

$E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ for fixed $k, n > 0$
where $\forall K \in \{0,1\}^k, E(K,\cdot)$ is a bijection

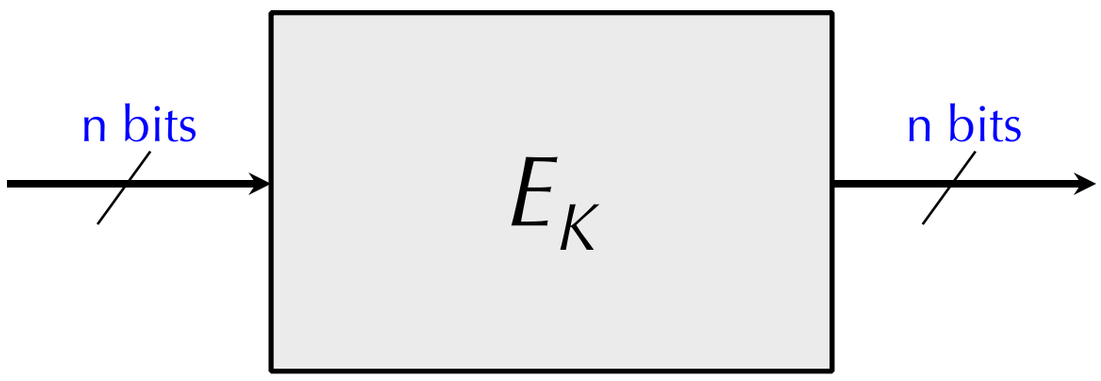
"function family" viewpoint

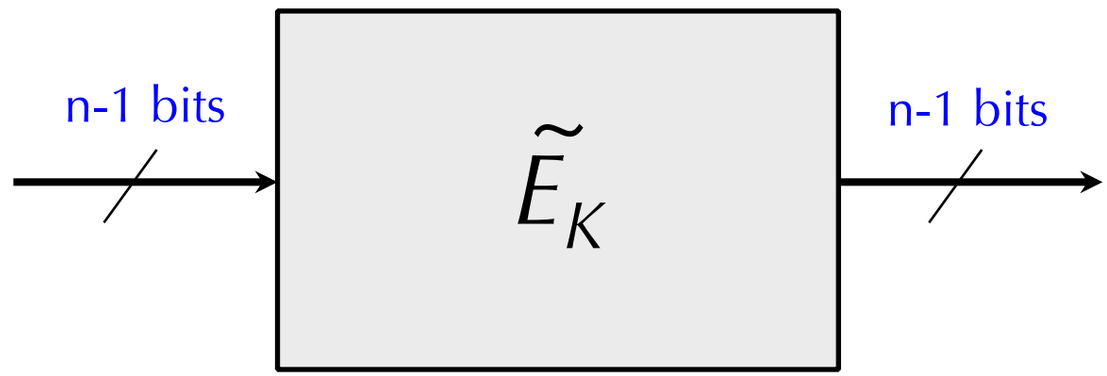
$\{E_K\colon \{0,1\}^n \to \{0,1\}^n \mid K \in \{0,1\}^k\}$
where $E_K \triangleq E(K,\cdot)$

PRP-security notion

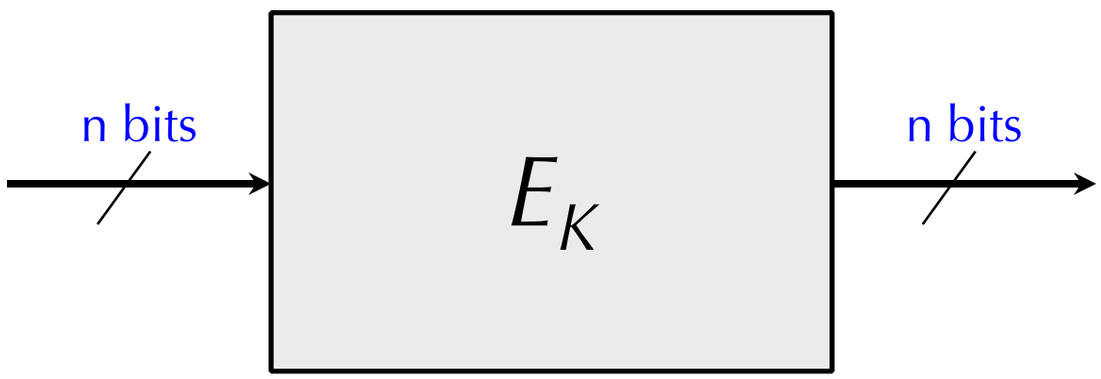


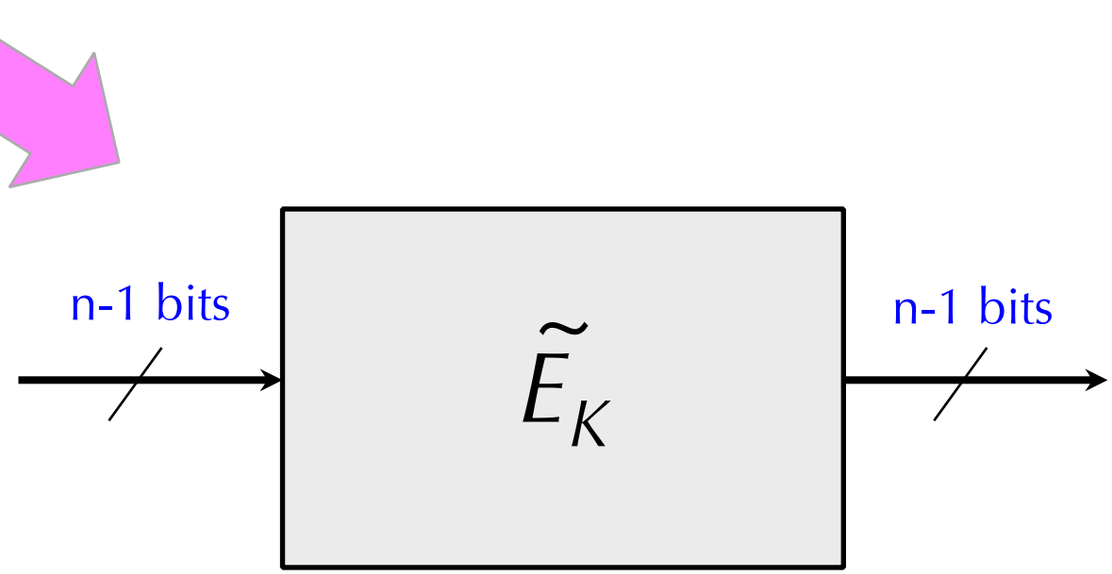
Turning an n-bit BC into an (n-1)-bit BC





Turning an n-bit BC into an (n-1)-bit BC

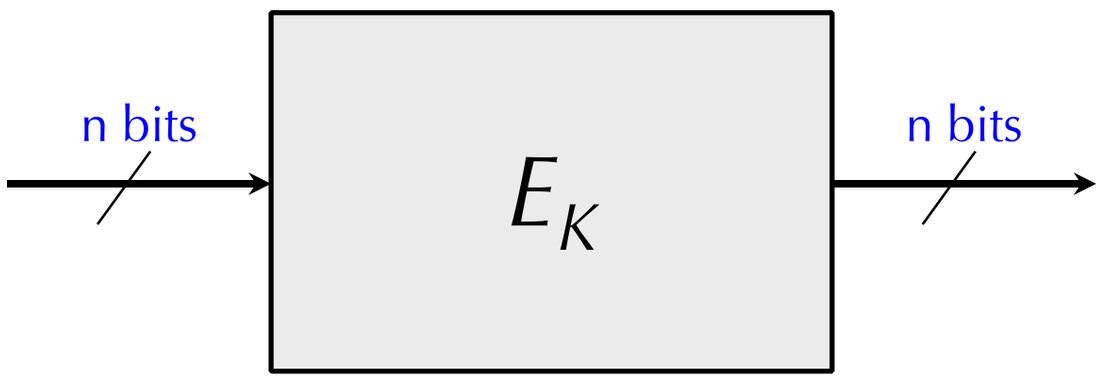


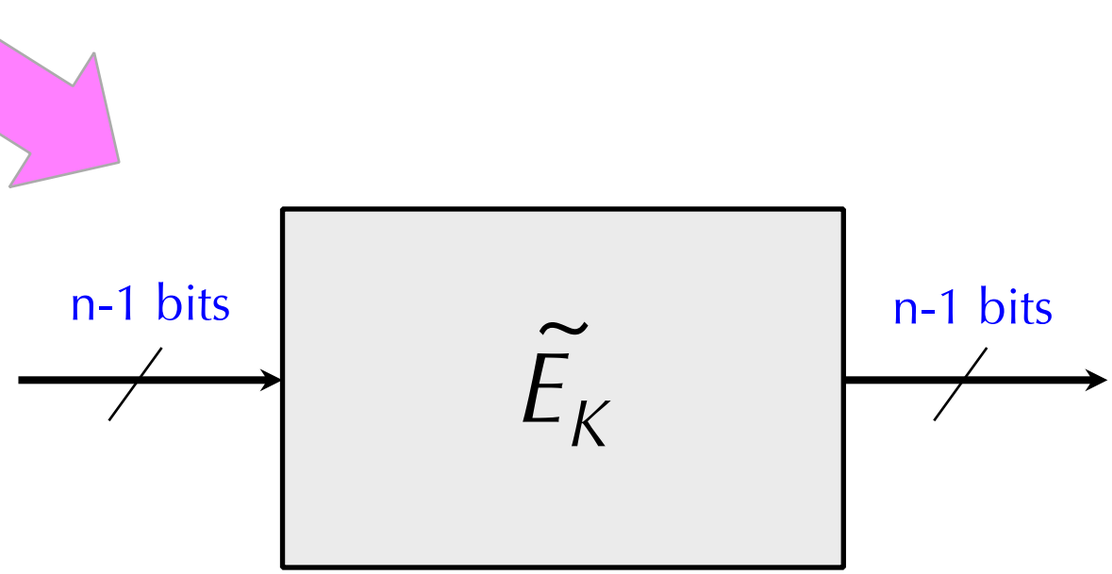


Idea #1: truncation!

$\tilde{E}_K(X) \triangleq (n-1)$ LSB of $E_K(X)$

Turning an n-bit BC into an (n-1)-bit BC

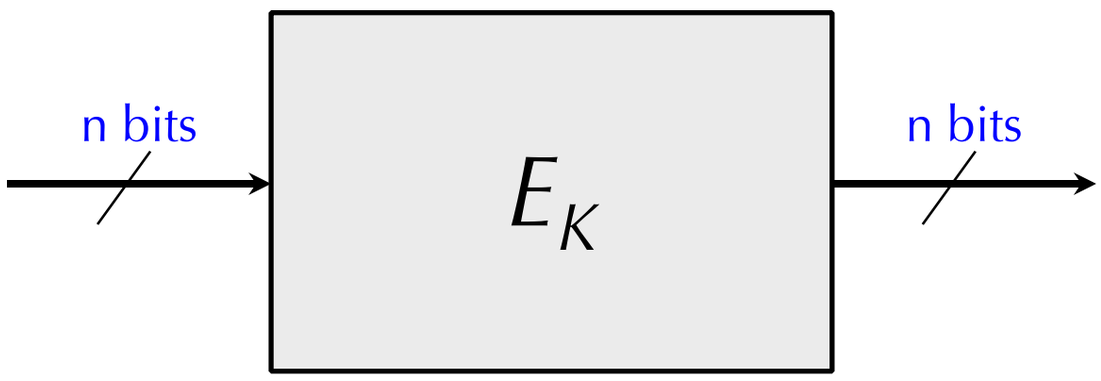


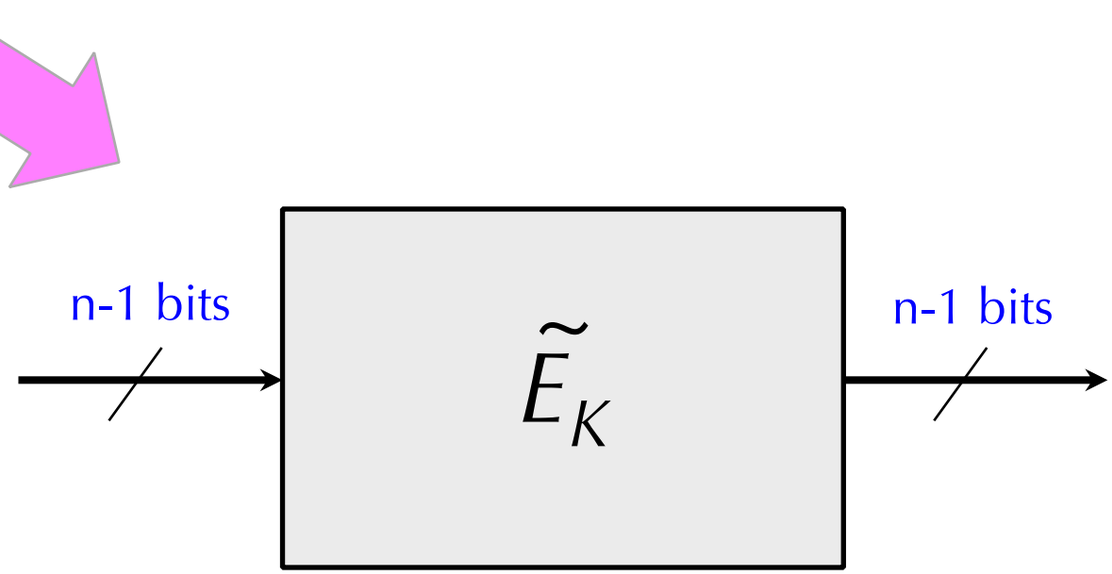


Idea #1: truncation!

$\tilde{E}_K(X) \triangleq (n-1)$ LSB of $E_K(0 \,||\, X)$

Turning an n-bit BC into an (n-1)-bit BC



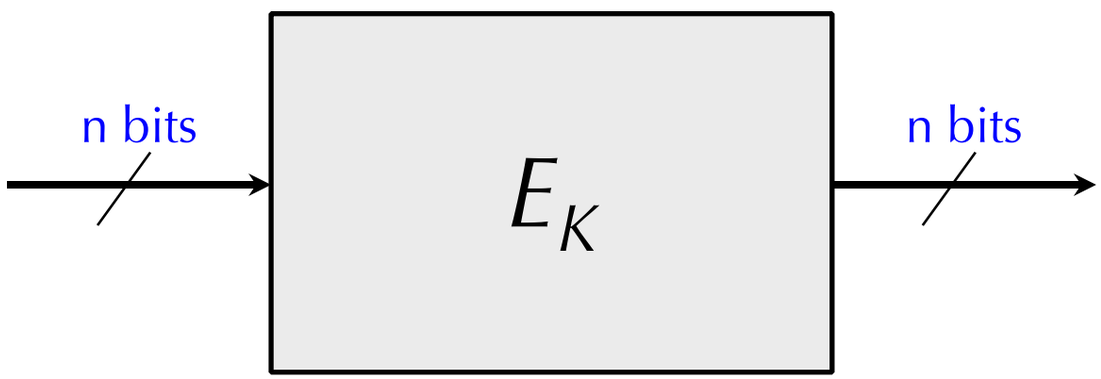


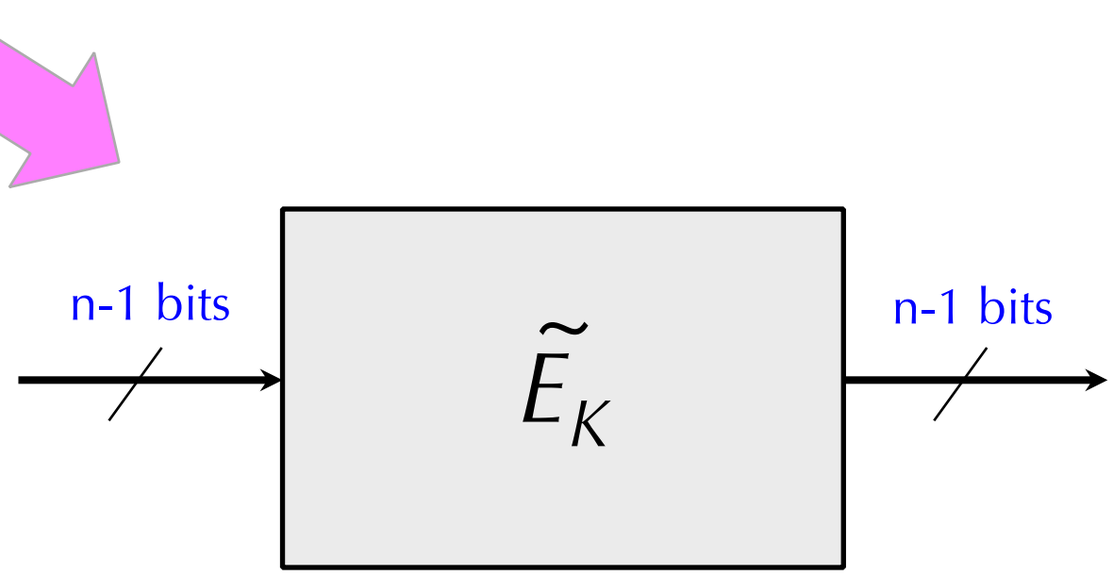
Idea #1: truncation!

$\tilde{E}_K(X) \triangleq (n-1)$ LSB of $E_K(0 \,||\, X)$

$\tilde{E}_K^{-1}(Y) \triangleq \ldots$ **How do we invert?**

Turning an n-bit BC into an (n-1)-bit BC



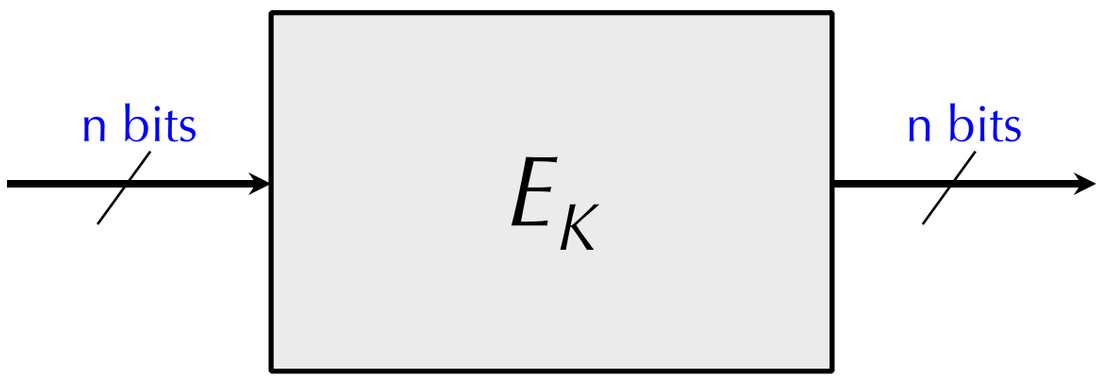


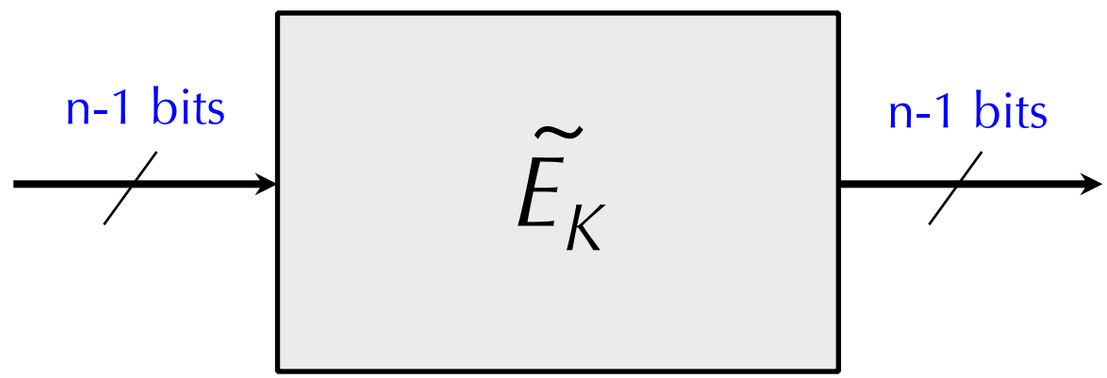
Idea #1: truncation!

$\tilde{E}_K(X) \triangleq (n-1)$ LSB of $E_K(0 \,||\, X)$

$\tilde{E}_K^{-1}(Y) \triangleq$ if $E_K^{-1}(0 \,\|\, Y) = 0 \,\|\, X$, return X; otherwise $E_K^{-1}(1 \,\|\, Y) = 0 \,\|\, X$, return X

Turning an n-bit BC into an (n-1)-bit BC





Idea #1: truncation!

$\tilde{E}_K(X) \triangleq (n-1)$ LSB of $E_K(0 \,||\, X)$

$\tilde{E}_K^{-1}(Y) \triangleq$ if $E_K^{-1}(0 \,\|\, Y) = 0 \,\|\, X$, return X; otherwise $E_K^{-1}(1 \,\|\, Y) = 0 \,\|\, X$, return X

Does it work?!

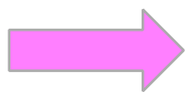
Turning an n-bit BC into an (n-1)-bit BC

$$\tilde{E}_K(X) \triangleq (n-1) \text{ LSB of } E_K(0 \,||\, X)$$

$$\tilde{E}_K^{-1}(Y) \triangleq \text{ if } E_K^{-1}(0 \,\|\, Y) = 0 \,\|\, X, \text{ return } X; \text{ otherwise } E_K^{-1}(1 \,\|\, Y) = 0 \,\|\, X, \text{ return } X$$

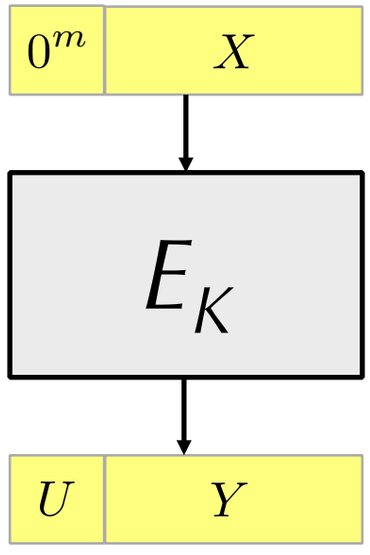
Let's consider a simple example, n=2

$$E_K: \begin{aligned} 00 &\mapsto 01 \\ 01 &\mapsto 11 \\ 10 &\mapsto 10 \\ 11 &\mapsto 00 \end{aligned}$$



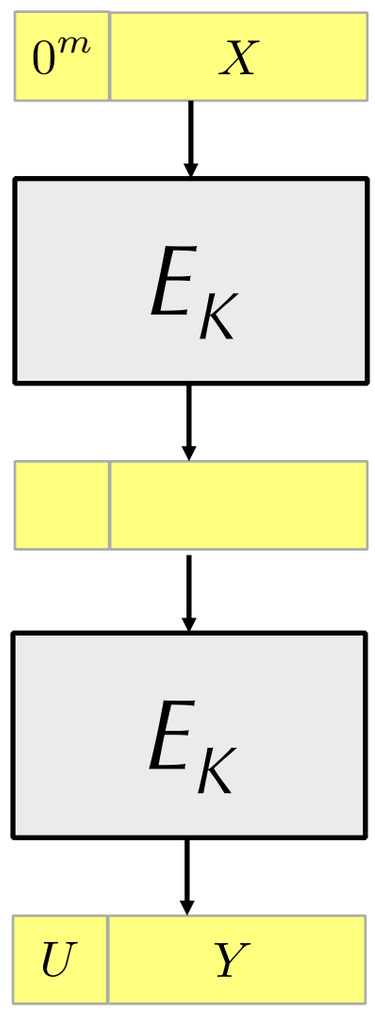
$$\tilde{E}_K: \begin{aligned} 0 &\mapsto 1 \\ 1 &\mapsto 1 \end{aligned}$$

Turning an n-bit BC into an (n-m)-bit BC



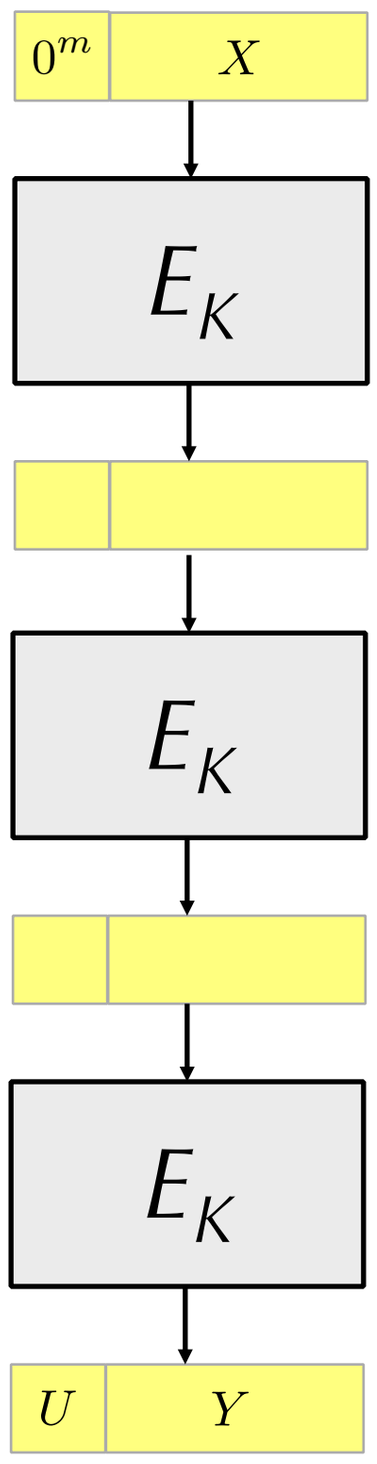
If U=0^m then return Y

Turning an n-bit BC into an (n-m)-bit BC



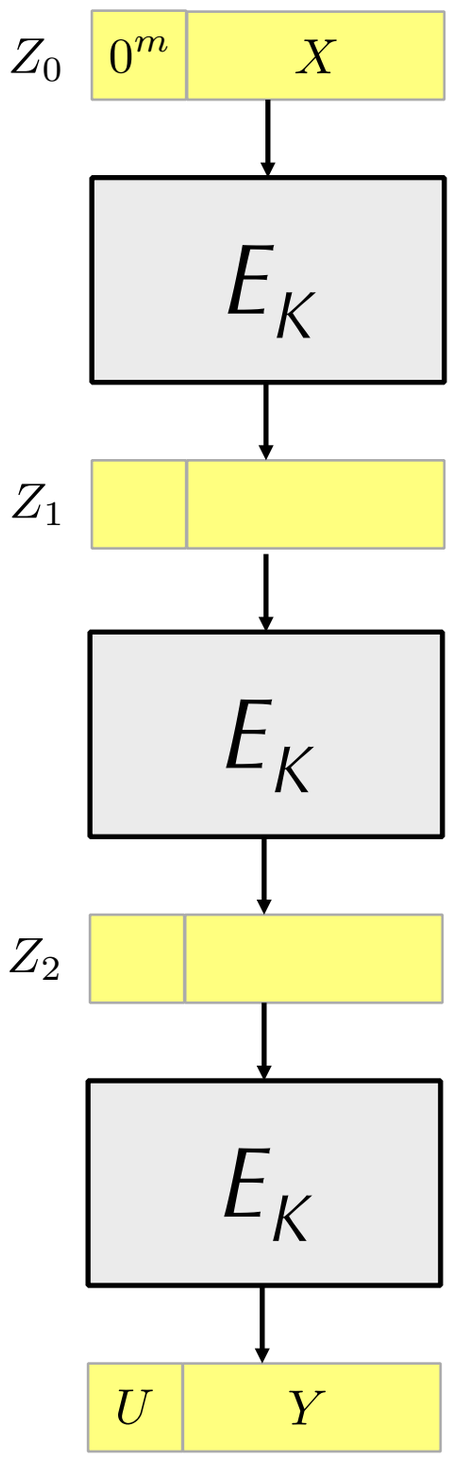
If U=0^m then return Y

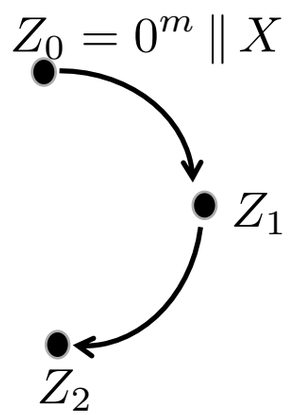
Turning an n-bit BC into an (n-m)-bit BC



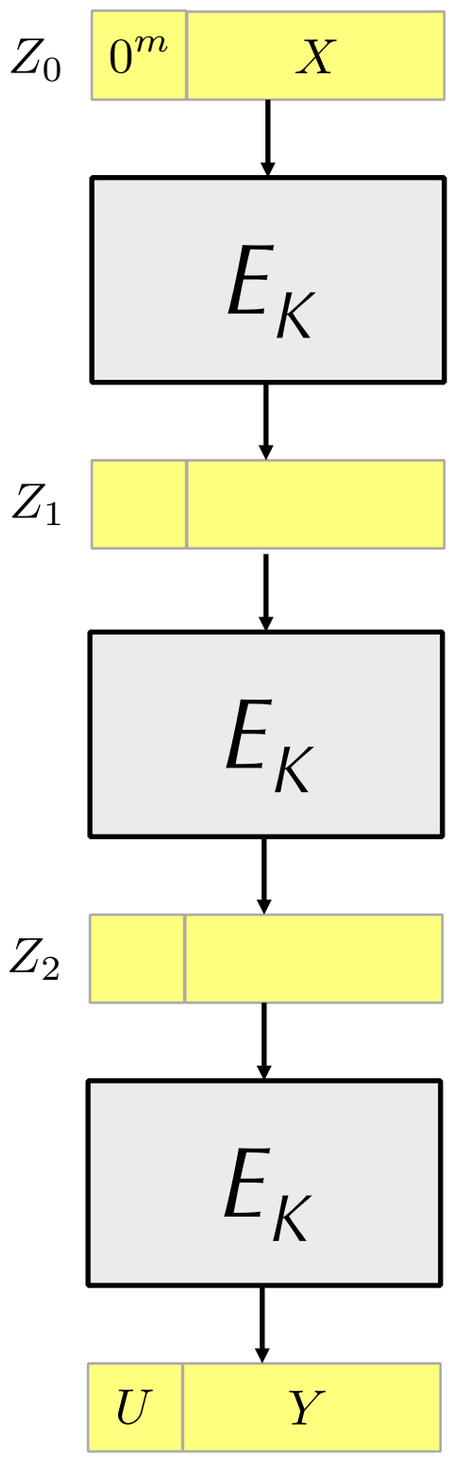
If U=0^m then return Y

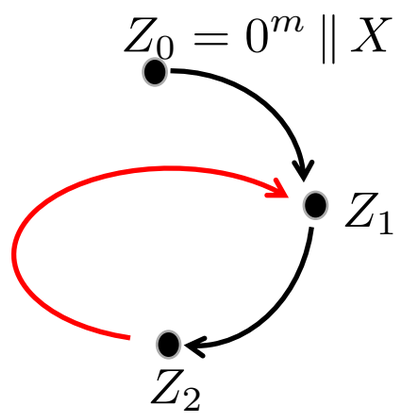
"Cycle Walking"



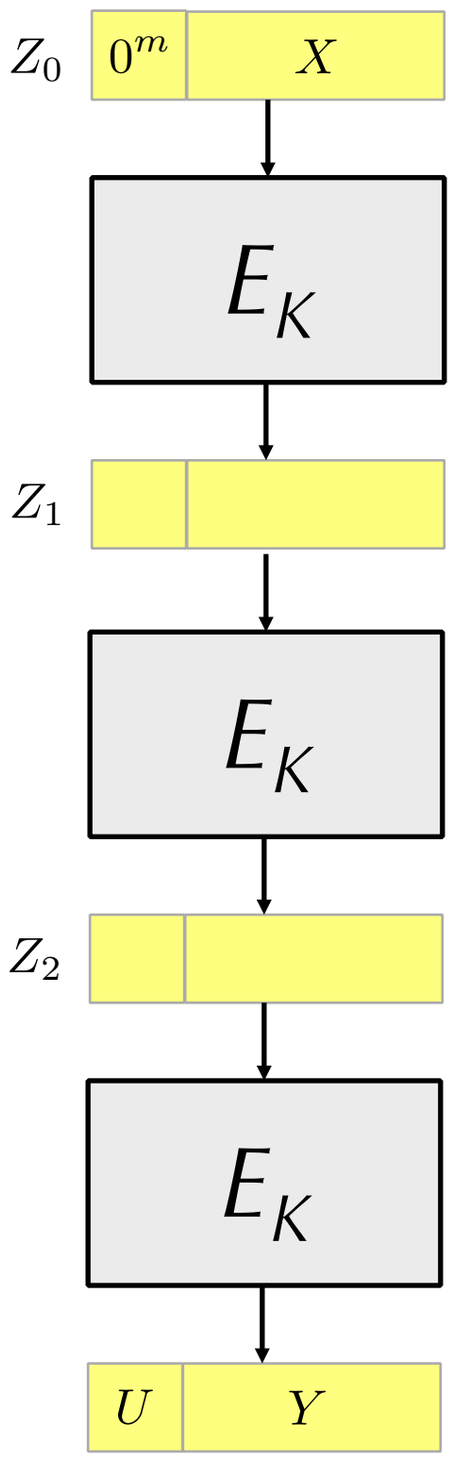


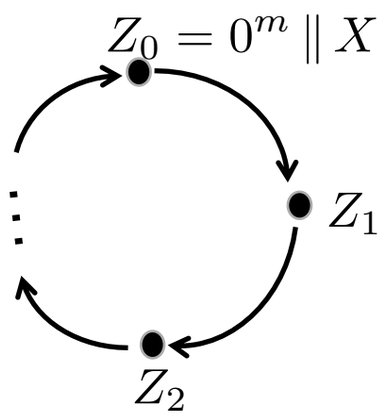
"Cycle Walking"





"Cycle Walking"





Expected # of steps $\leq 2^m$

"Cycle Walking"

Expected # of steps $\leq 2^m$ (in fact, Chernoff-like bounds show #steps close to expectation with overwhelming probability)

Security? $\mathbf{Adv}^{\mathrm{prp}}_{\tilde{E}}(t, q) \leq \mathbf{Adv}^{\mathrm{prp}}_{E}(t + O(q2^m), q2^m)$ (with overwhelming probability)

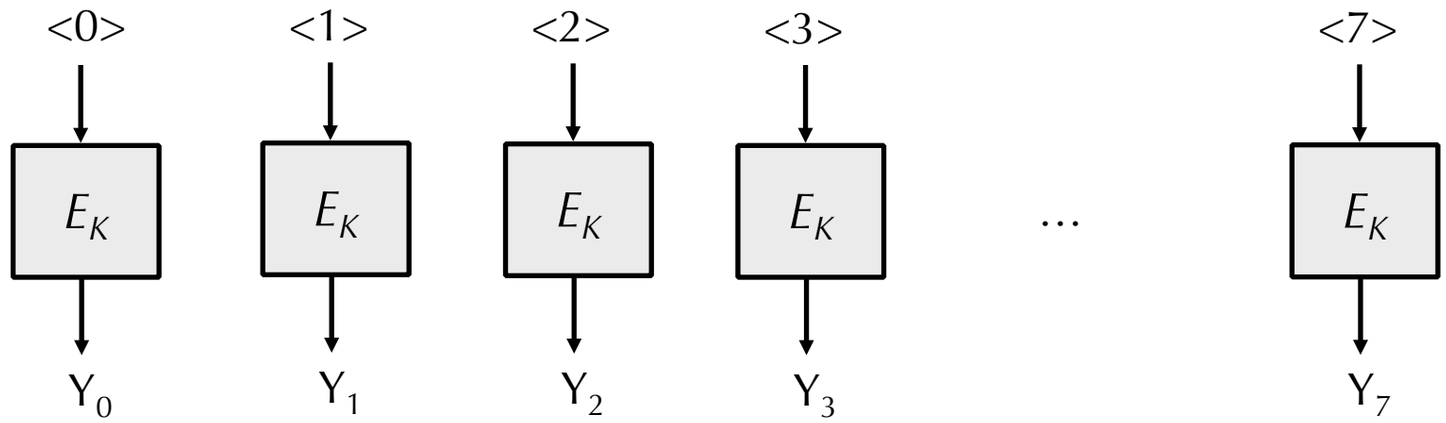
Clearly, this method is useful only if m is small, i.e. (n-m) is "close" to n

What about when you want to shrink the domain *a lot*?

Say you want a 3-bit BC

<*k*> = encoding of integer *k* as an **n**-bit string

1. Encipher the entire 3-bit domain

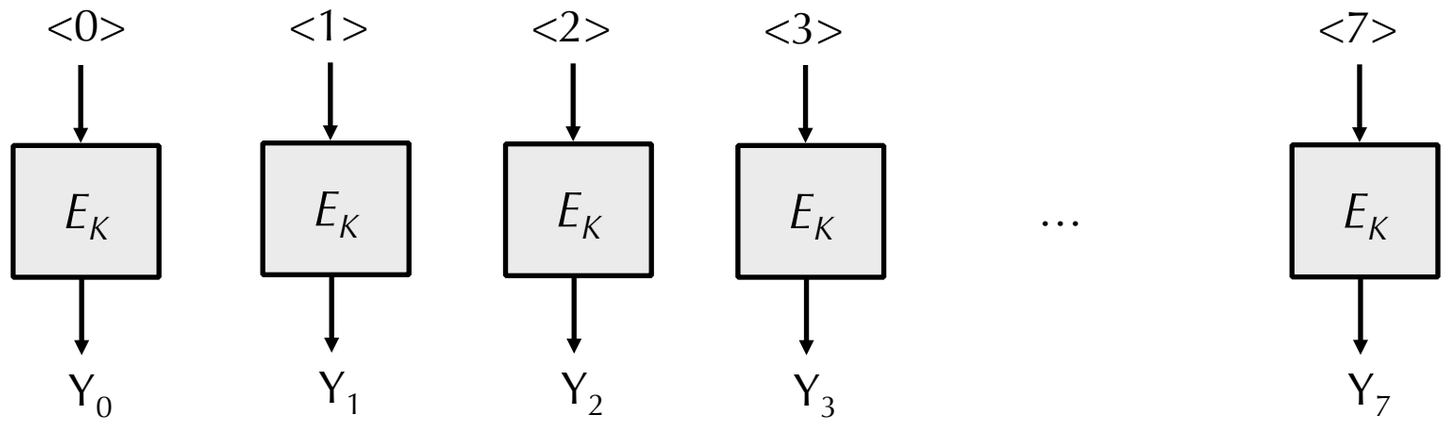


What about when you want to shrink the domain *a lot*?

Say you want a 3-bit BC

<*k*> = encoding of integer *k* as an n-bit string

1. Encipher the entire 3-bit domain



2. Sort the resulting n-bit values

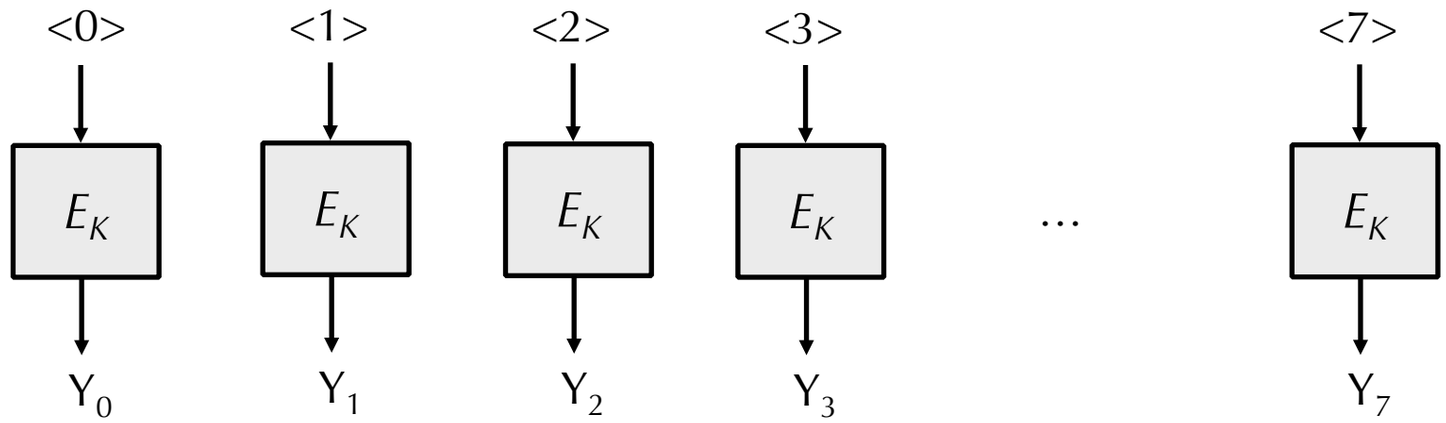
$$Y_2 < Y_7 < Y_5 < Y_3 < Y_0 < Y_1 < Y_4 < Y_6$$

What about when you want to shrink the domain *a lot*?

Say you want a 3-bit BC

<*k*> = encoding of integer *k* as an n-bit string

1. Encipher the entire 3-bit domain



2. Sort the resulting n-bit values

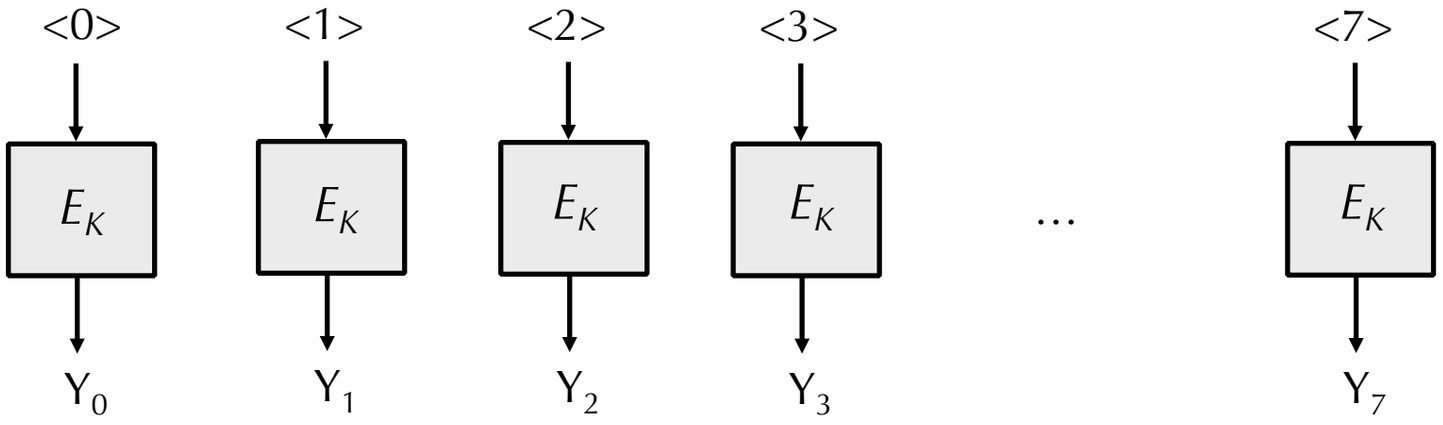
$$Y_2 < Y_7 < Y_5 < Y_3 < Y_0 < Y_1 < Y_4 < Y_6$$

3. Define the mapping "index to position"

$$\tilde{E}_K(\langle 2\rangle) = \langle 0\rangle,\ \tilde{E}_K(\langle 7\rangle) = \langle 1\rangle,\ \tilde{E}_K(\langle 5\rangle) = \langle 2\rangle,\ \ldots,\ \tilde{E}_K(\langle 6\rangle) = \langle 7\rangle$$

Best possible security!

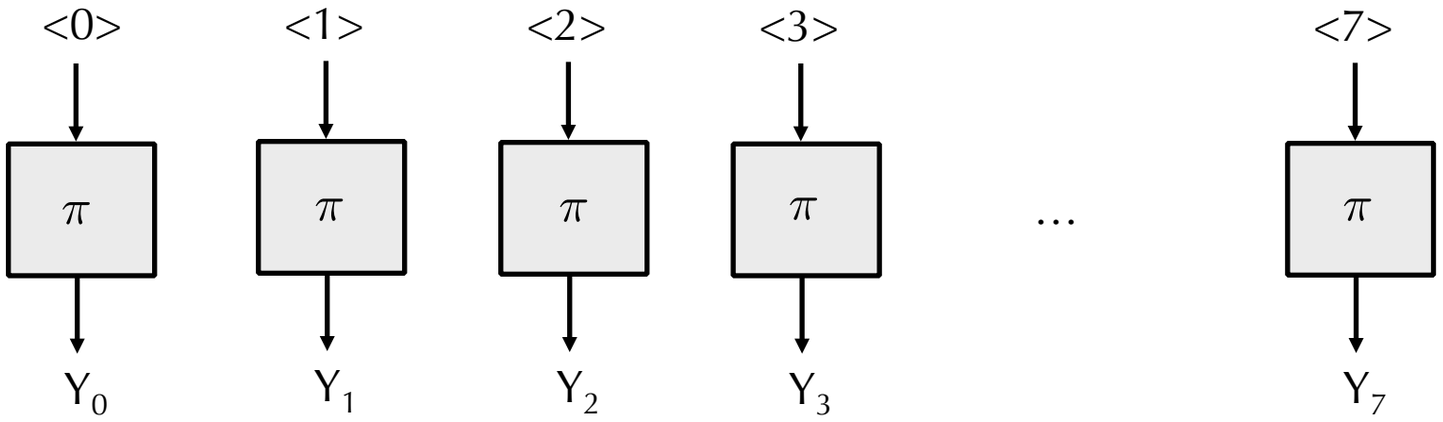
<k> = encoding of integer
k as an n-bit string



$Y_2 < Y_7 < Y_5 < Y_3 < Y_0 < Y_1 < Y_4 < Y_6$

Best possible security!

<k> = encoding of integer
k as an n-bit string



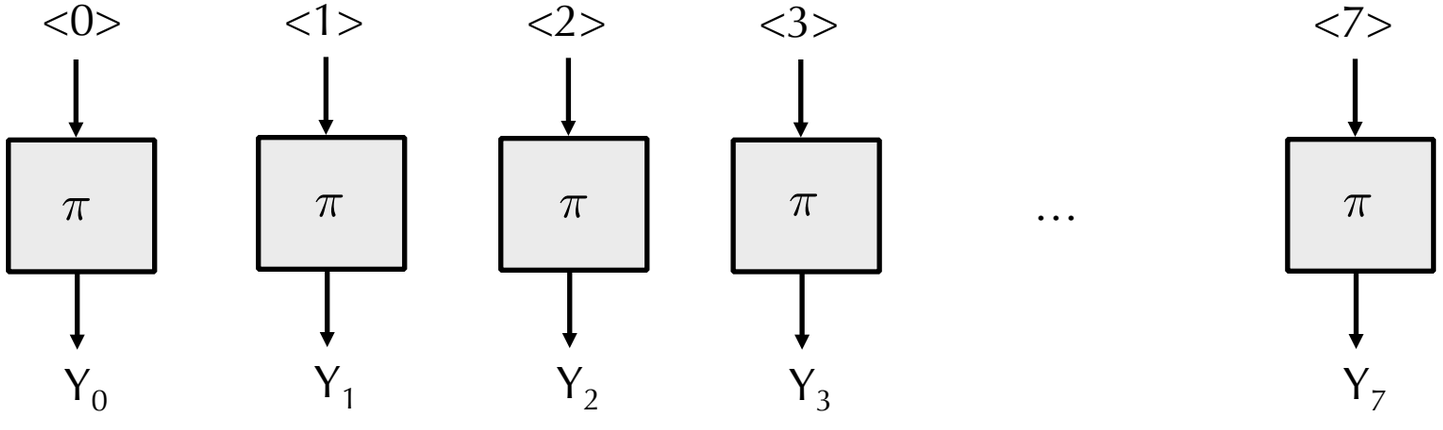
$Y_2 < Y_7 < Y_5 < Y_3 < Y_0 < Y_1 < Y_4 < Y_6$

Y_2 **Y_7** **Y_5** **Y_4** **Y_6**

$$\Pr\left[\pi \leftarrow\!\!\$\ \mathrm{Perm}\,(n)\ :\ \pi(\langle 2\rangle) < \pi(\langle 7\rangle) < \pi(\langle 5\rangle) < \cdots < \pi(\langle 4\rangle) < \pi(\langle 6\rangle)\right] = ?$$

Best possible security!

<k> = encoding of integer
k as an n-bit string



$Y_2 < Y_7 < Y_5 < Y_3 < Y_0 < Y_1 < Y_4 < Y_6$

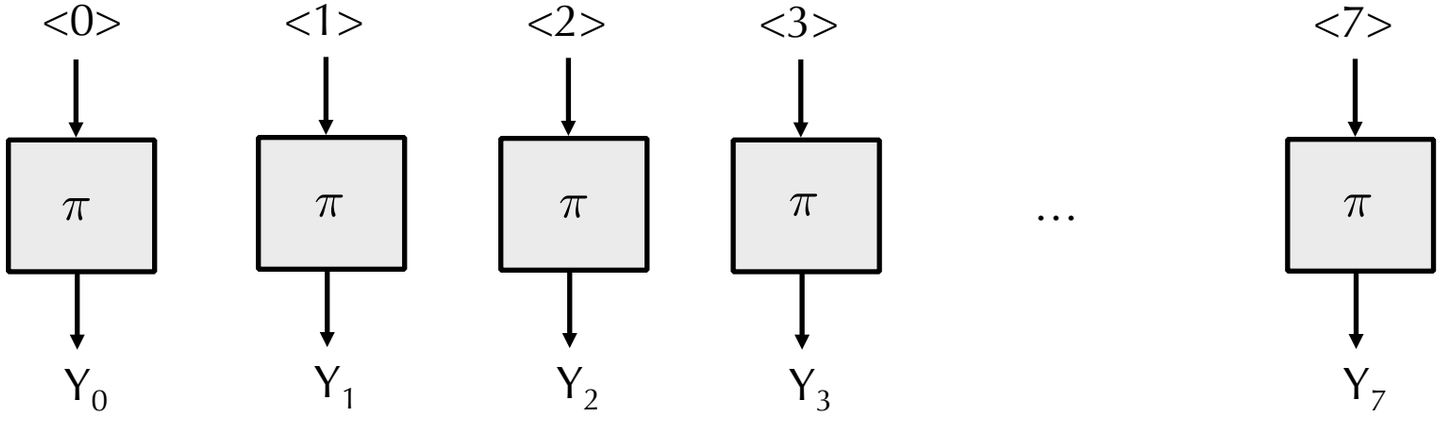
$$\Pr\left[\pi \leftarrow\!\!\$\ \mathrm{Perm}\,(n)\ :\ \underset{\mathbf{Y_2}}{\pi(\langle 2\rangle)} < \underset{\mathbf{Y_7}}{\pi(\langle 7\rangle)} < \underset{\mathbf{Y_5}}{\pi(\langle 5\rangle)} < \cdots < \underset{\mathbf{Y_4}}{\pi(\langle 4\rangle)} < \underset{\mathbf{Y_6}}{\pi(\langle 6\rangle)}\right] = \frac{\binom{2^n}{8}\cdot(2^n-8)!}{(2^n)!}$$

$$= \frac{(2^n)!\cdot(2^n-8)!}{8!(2^n-8)!(2^n)!}$$

$$= \frac{1}{8!}$$

Best possible security!

<k> = encoding of integer
k as an n-bit string



$Y_2 < Y_7 < Y_5 < Y_3 < Y_0 < Y_1 < Y_4 < Y_6$

$$\Pr\left[\pi \leftarrow\!\!\$\ \mathrm{Perm}\,(n)\ :\ \underset{Y_2}{\pi(\langle 2\rangle)} < \underset{Y_7}{\pi(\langle 7\rangle)} < \underset{Y_5}{\pi(\langle 5\rangle)} < \cdots < \underset{Y_4}{\pi(\langle 4\rangle)} < \underset{Y_6}{\pi(\langle 6\rangle)}\right] = \frac{\binom{2^n}{8}\cdot(2^n-8)!}{(2^n)!}$$

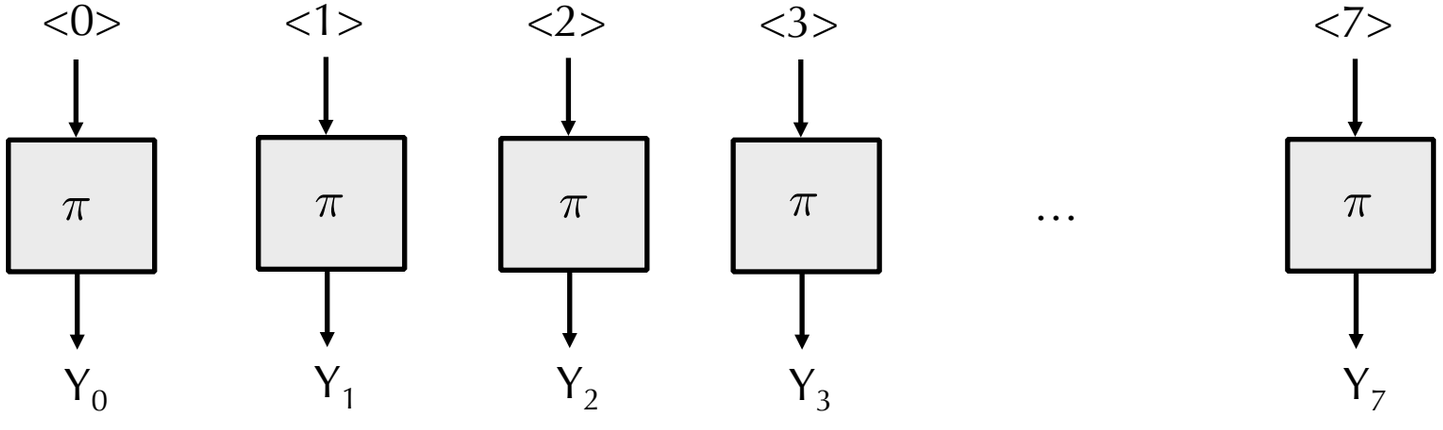
$$= \frac{(2^n)!\cdot(2^n-8)!}{8!(2^n-8)!(2^n)!}$$

$$= \frac{1}{8!}$$

Recall that our *defined* BC is over $\{0,1\}^3$

Best possible security!

<k> = encoding of integer
k as an n-bit string



$Y_2 < Y_7 < Y_5 < Y_3 < Y_0 < Y_1 < Y_4 < Y_6$

$$\Pr[\pi \leftarrow\!\$\ \mathrm{Perm}(n) \ :\ \pi(\langle 2\rangle) < \pi(\langle 7\rangle) < \pi(\langle 5\rangle) < \cdots < \pi(\langle 4\rangle) < \pi(\langle 6\rangle)] = \frac{\binom{2^n}{8}\cdot(2^n-8)!}{(2^n)!}$$

$$= \frac{(2^n)!\cdot(2^n-8)!}{8!(2^n-8)!(2^n)!}$$

$$= \frac{1}{8!}$$

Recall that our *defined* BC is over $\{0,1\}^3$

$$\Pr[\tilde{\pi} \leftarrow\!\$\ \mathrm{Perm}(3) : \tilde{\pi}(\langle 2\rangle) < \tilde{\pi}(\langle 7\rangle) < \tilde{\pi}(\langle 5\rangle) < \cdots < \tilde{\pi}(\langle 4\rangle) < \tilde{\pi}(\langle 6\rangle)] = \frac{1}{8!}$$

"Prefix cipher"

Offline processing to build/store permutation,
fast online table-driven encryption

$$\mathbf{Adv}^{\mathrm{prp}}_{\tilde{E}}(t, q) \leq \mathbf{Adv}^{\mathrm{prp}}_{E}(t + O(2^b) + O(qb), 2^b)$$

Only useful if the target domain is tiny compared to $\{0,1\}^n$

"Cycle Walking"

Expected # of steps $\leq 2^m$
(with overwhelming probability)

$$\mathbf{Adv}^{\mathrm{prp}}_{\tilde{E}}(t, q) \leq \mathbf{Adv}^{\mathrm{prp}}_{E}(t + O(q2^m), q2^m)$$

Only useful if the target domain is essentially $\{0,1\}^n$

What do we do for "mid-sized" target domains?!

An important "mid-sized" target domain

1234 5678 9876 5432 → Encrypt → 4417 1234 5678 9112

$\log_2(10^{16}) = 53.15$ bits to encode 16-digit decimal string

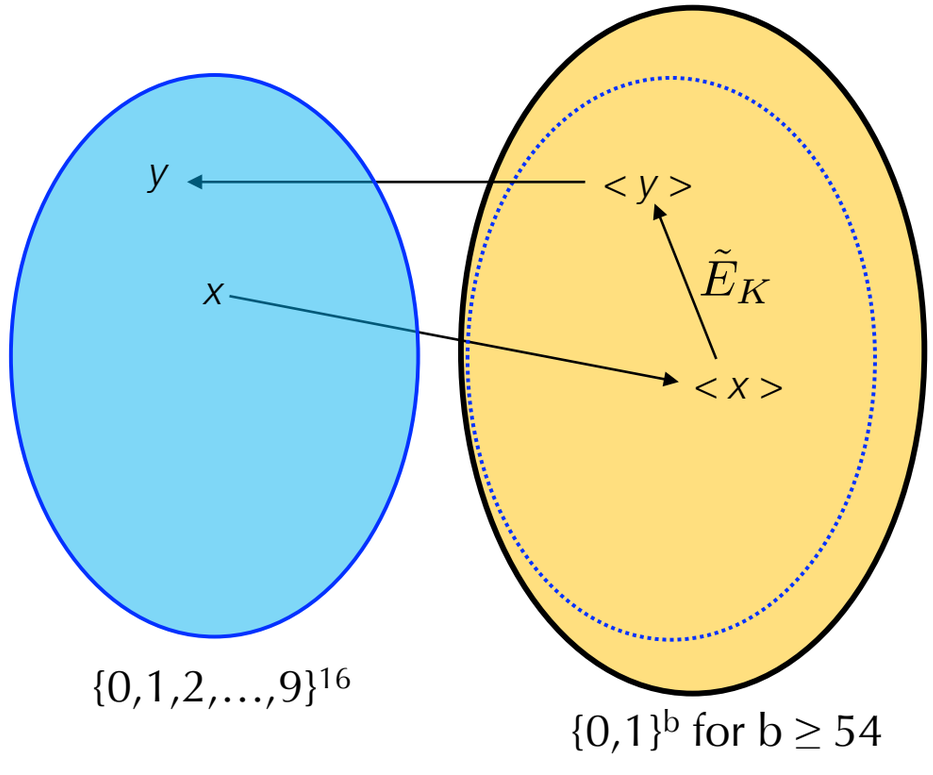
Need to build a 54-bit BC from (say) AES

"Prefix cipher"
$O(2^{54})$ storage and precomputation

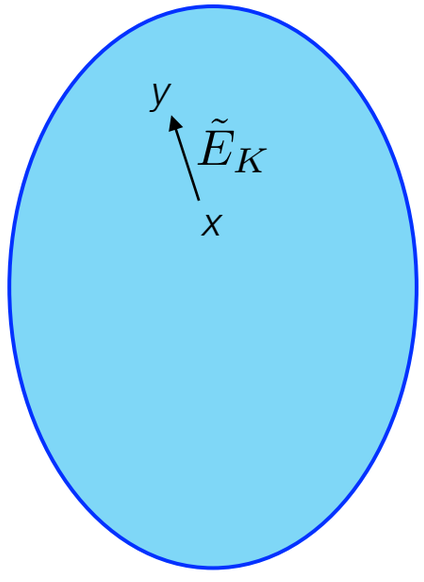
"Cycle Walking"
$O(2^{74})$ steps for a single encryption

Let's take a step back...

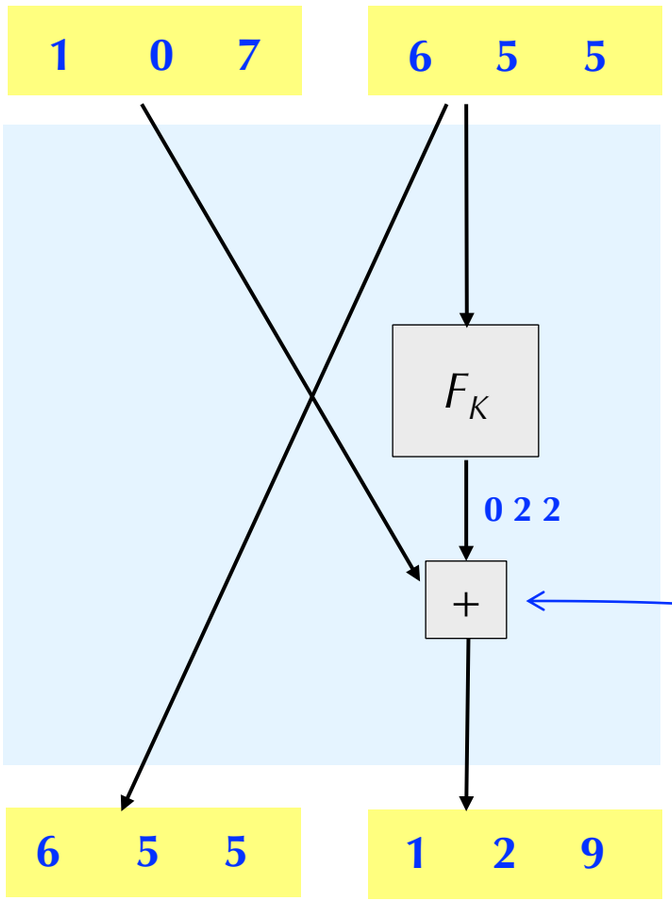
Do we need to convert from $\{0,1,2,\ldots,9\}^{16}$ to bitstrings?



What if we could encipher $\{0,1,\ldots,9\}^{16}$ **directly** (more or less)?



One round of “Integer Feistel”



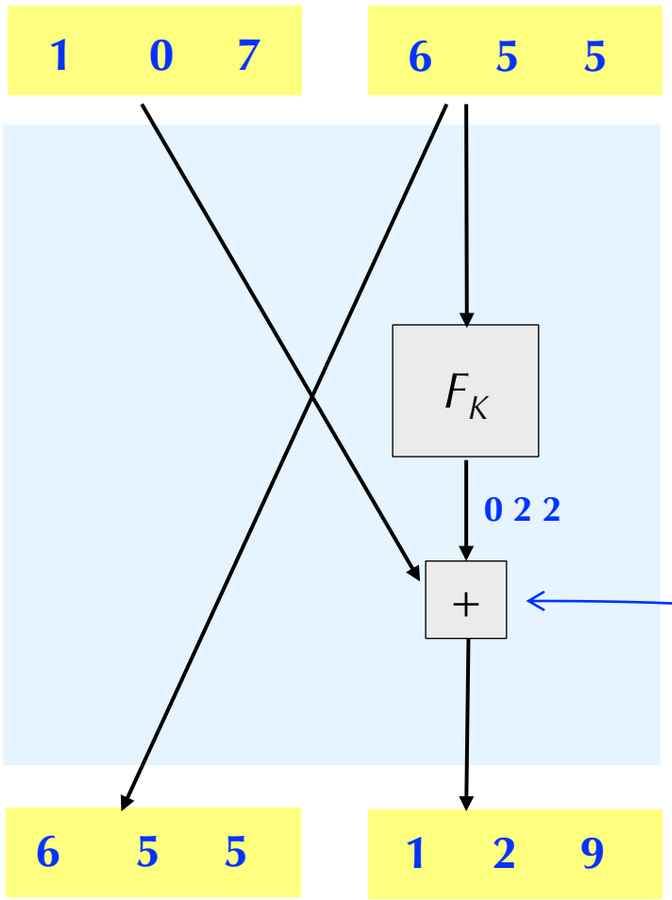
Round function $F_K : \{0,1,\ldots,9\}^* \to \{0,1,\ldots,9\}^3$

symbol-wise addition mod 10

$$\tilde{E}_K^{(1)}(107655) = 655129$$

A cipher for 6-digit decimal strings!

One round of "Integer Feistel"



Round function $F_K : \{0,1,\ldots,9\}^* \to \{0,1,\ldots,9\}^3$

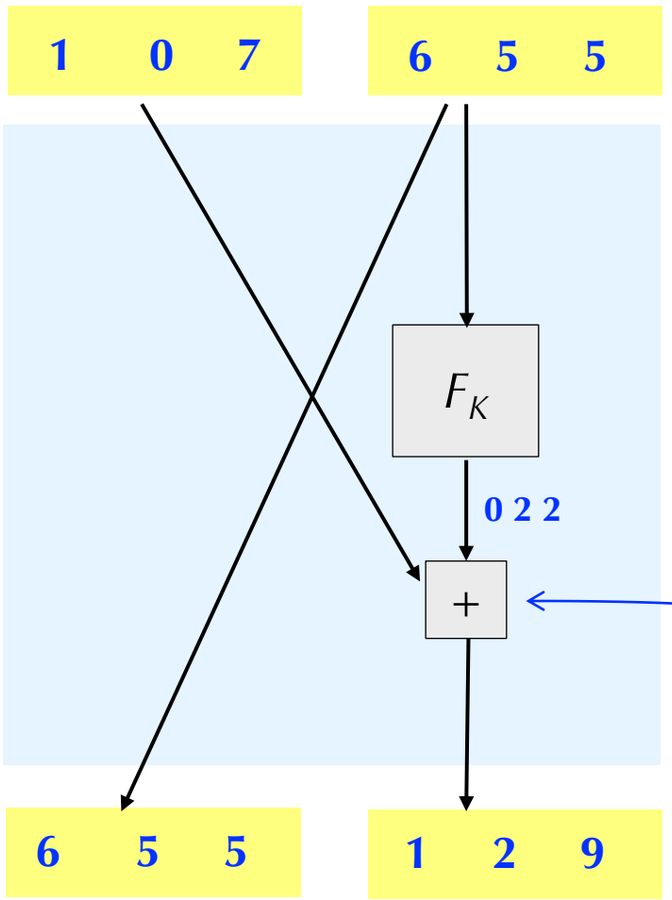
e.g. **F_K(N) = AES-CBCMAC$_K$(<N>) truncated to ceiling(log$_2$(1000)) bits, taken mod 1000**

symbol-wise addition mod 10

$$\tilde{E}_K^{(1)}(107655) = 655129$$

A cipher for 6-digit decimal strings!

One round of "Integer Feistel"



Round function $F_K : \{0,1,\ldots,9\}^* \to \{0,1,\ldots,9\}^3$

e.g. **F_K(N) = AES-CBCMAC$_K$(<N>) truncated to ceiling($\log_2$(1000)) bits, taken mod 1000**

symbol-wise addition mod 10

$$\tilde{E}_K^{(1)}(107655) = 655129$$

A cipher for 6-digit decimal strings!

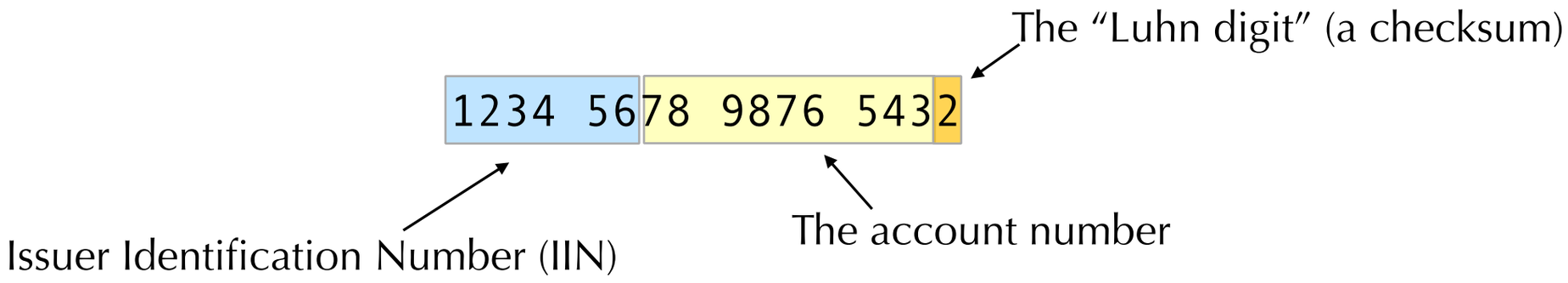
(See recent papers on "shuffling" by Rogaway and others)

Notes:

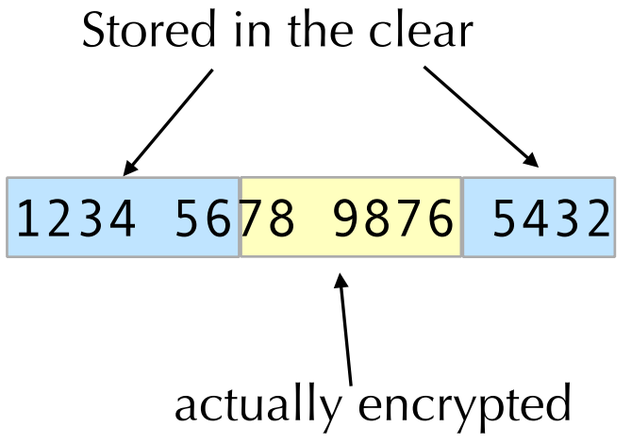
(1) need multiple (many?) rounds to get good provable security

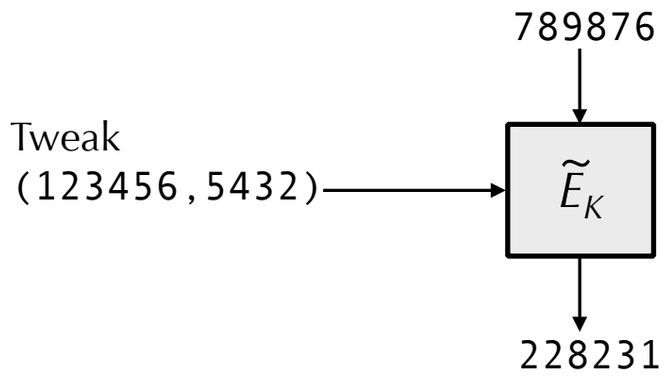
(2) easy to make this a tweakable cipher by throwing the tweak (and maybe the round number) into the F_K computation

Encrypting CCs



Encrypting CCs



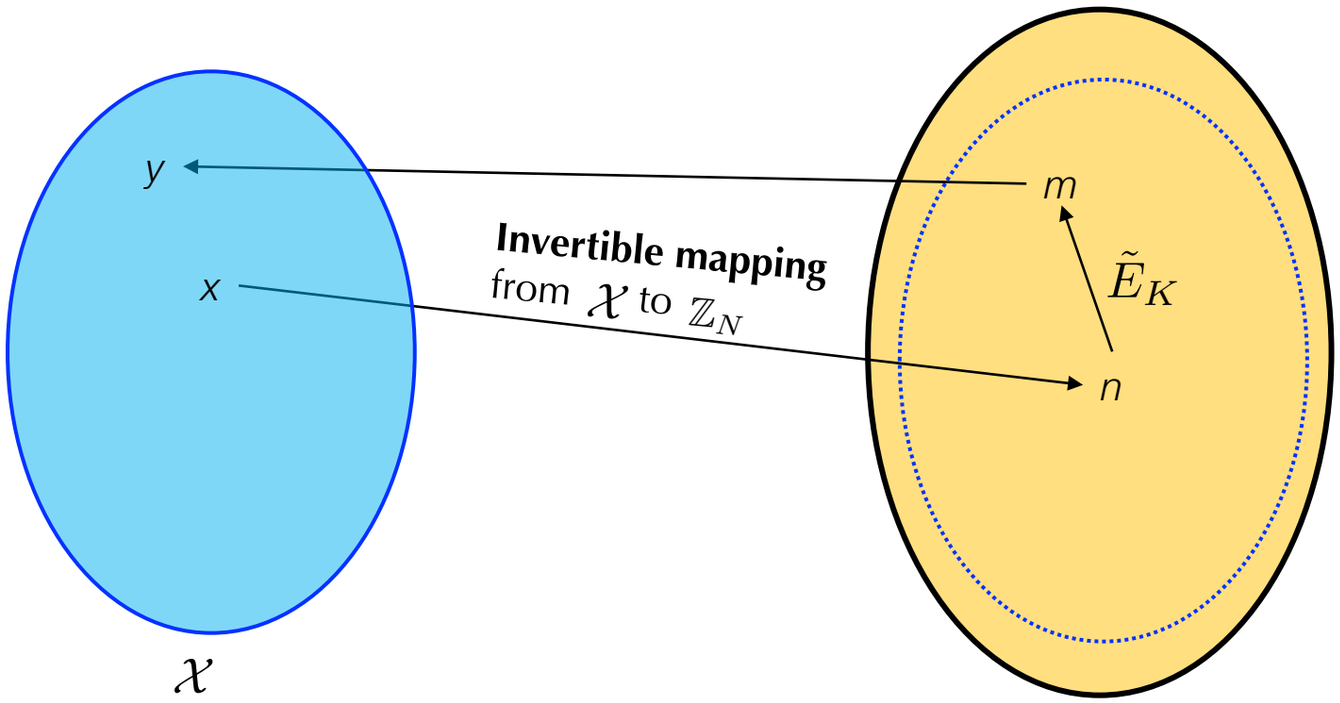


e.g. tweaked integer Feistel

1234 5622 8231 5432 **Encrypted CC**

Now that we have a integer cipher...

$$\tilde{E}\colon \mathcal{K} \times \mathbb{Z}_N \to \mathbb{Z}_N$$

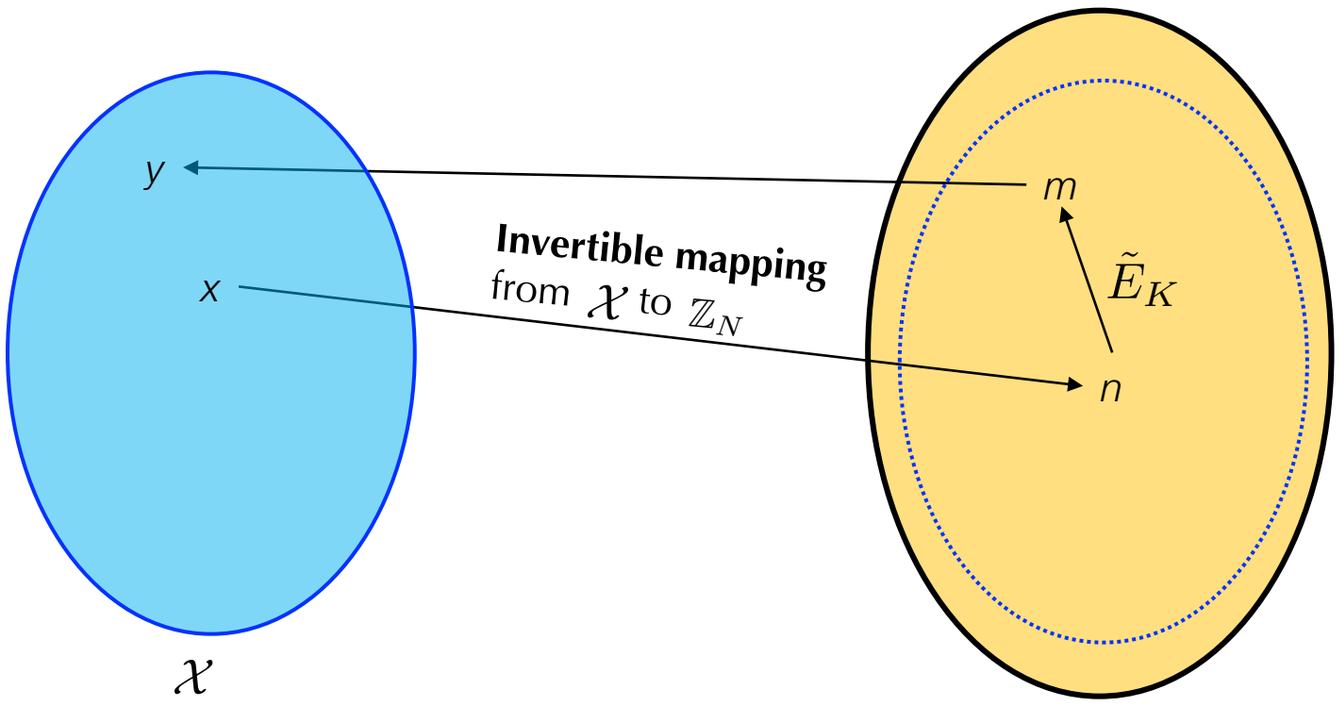


(what if m is "outside" the image of $\mathcal{X}$?)

$\tilde{E}_K$ **+ invertible mapping** ➡ $\tilde{F}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ **a cipher over** $\mathcal{X}$ **!**

Format-preserving encryption (FPE) [Bellare et al. '09]

$$\tilde{E}\colon \mathcal{K} \times \mathbb{Z}_N \to \mathbb{Z}_N$$



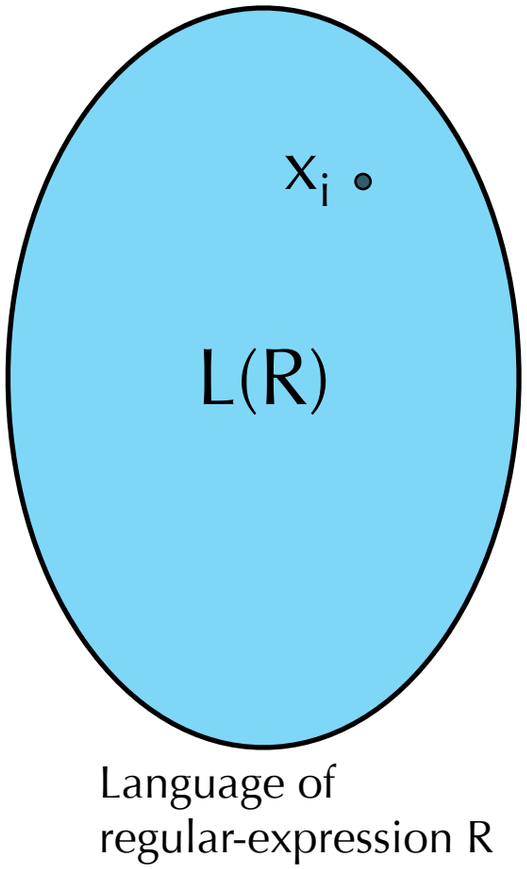
(what if m is "outside" the image of $\mathcal{X}$?)

$\tilde{E}_K$ + **invertible mapping** ➡ $\tilde{F}\colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ **a cipher over $\mathcal{X}$!**

How to do this?
How to do it *efficiently*?!

Ranking a Regular Language

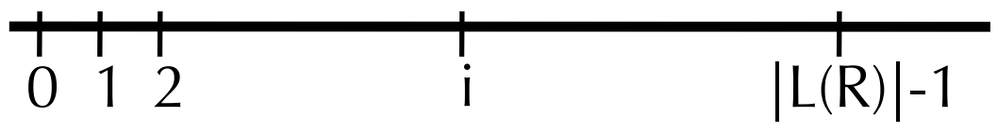
[Goldberg, Sipser '85]
[Bellare et al. '09]



Language of
regular-expression R

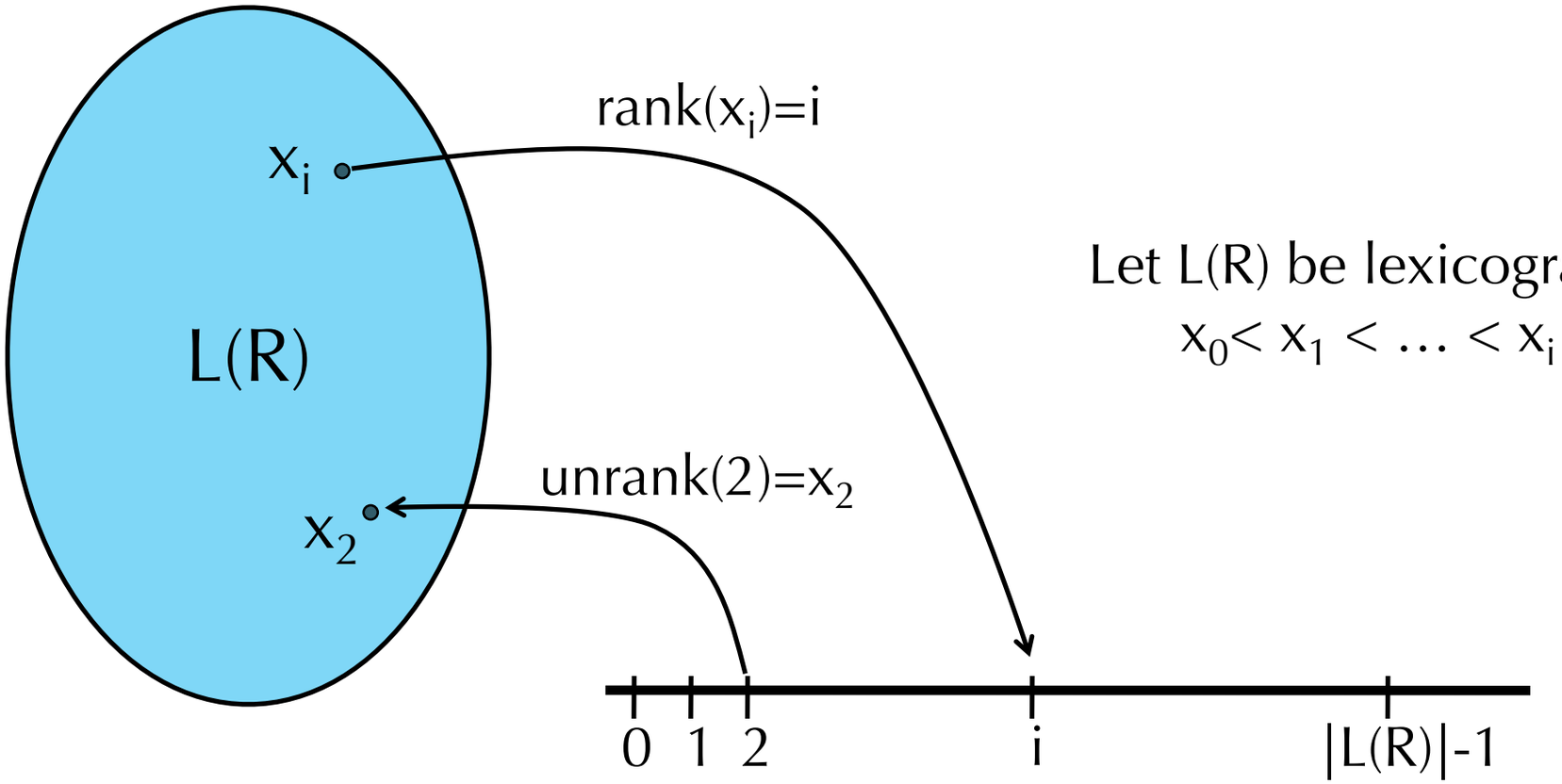
Let L(R) be lexicographically ordered

$x_0 < x_1 < \ldots < x_i < \ldots < x_{|L(R)-1|}$



Ranking a Regular Language

[Goldberg, Sipser '85]
[Bellare et al. '09]



Let L(R) be lexicographically ordered

$x_0 < x_1 < \ldots < x_i < \ldots < x_{|L(R)-1|}$

Given a **DFA** for L(R), there are efficient algorithms

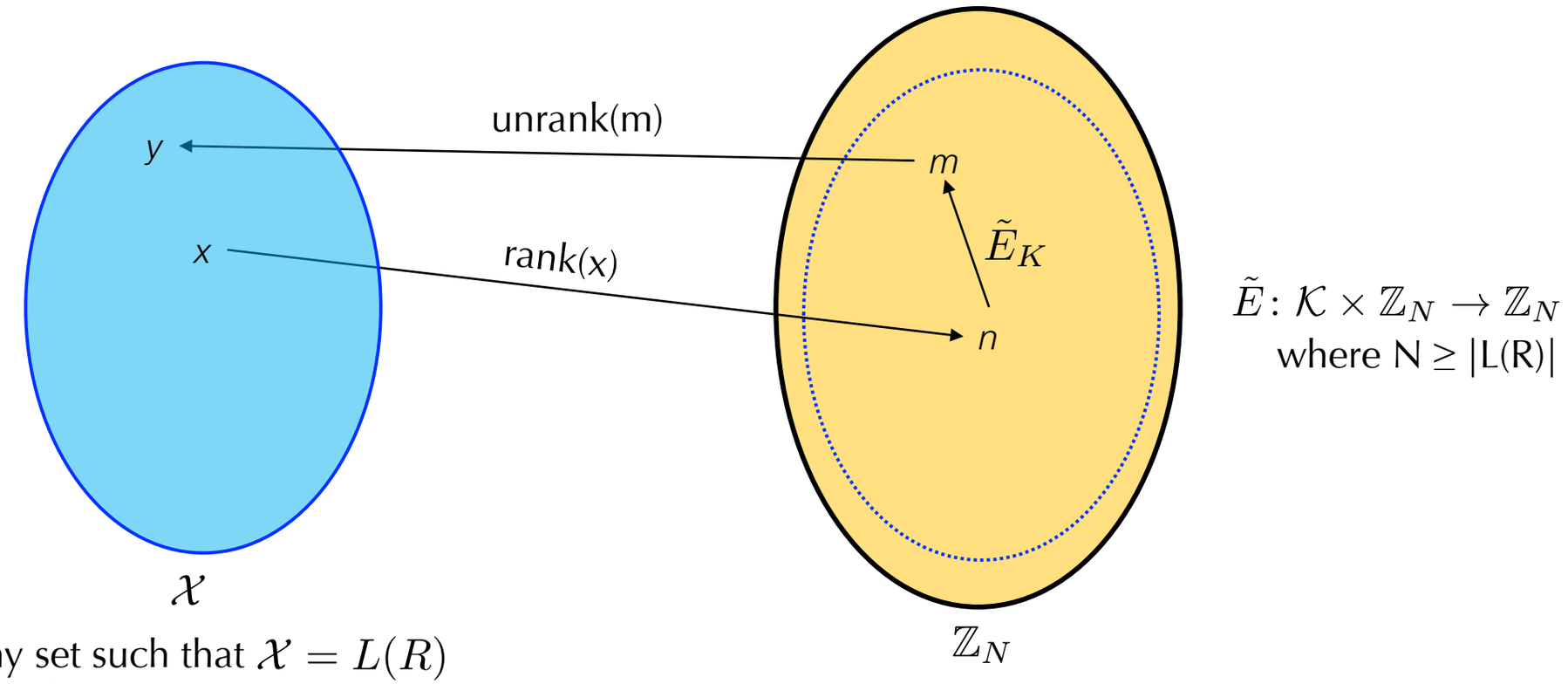
rank: L(R) $\longrightarrow$ {0,1,…,|L(R)|-1}
unrank: {0,1,…,|L(R)|-1} $\longrightarrow$ L(R)

such that rank(unrank(i)) = i
and unrank(rank(x_i)) = x_i

With precomputed tables, rank, unrank are *O*(n)

Regular-expression based FPE [Bellare et al. '09]

$$\tilde{F}\colon \mathcal{K} \times L(R) \to L(R)$$

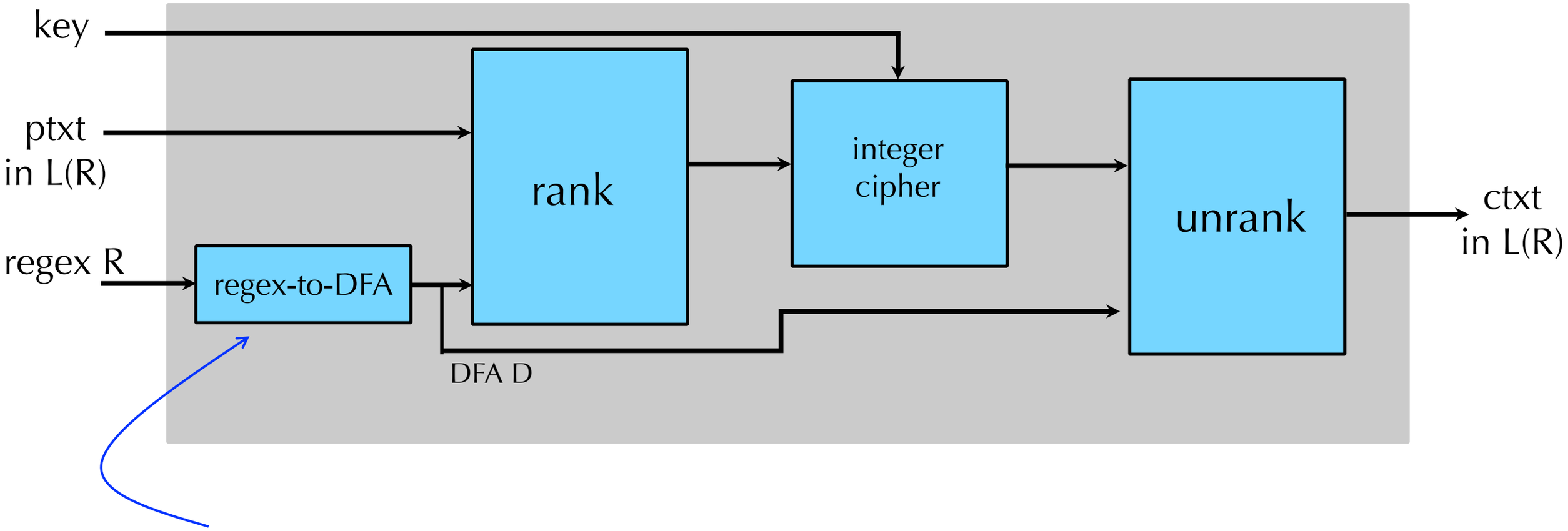


$\tilde{E}\colon \mathcal{K} \times \mathbb{Z}_N \to \mathbb{Z}_N$
where N ≥ |L(R)|

any set such that $\mathcal{X} = L(R)$
for a known regex *R*

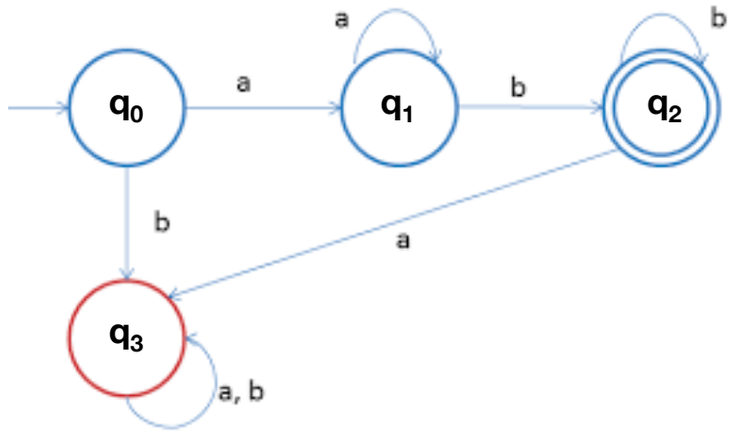
“rank-encrypt-unrank” FPE construction

[Bellare et al. ’09]



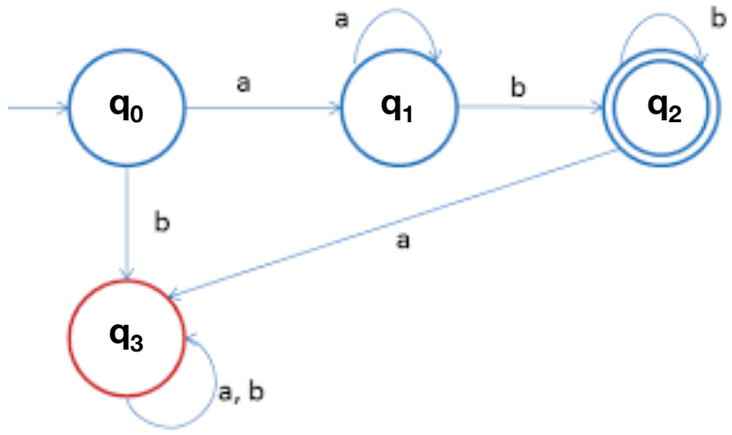
Why do we need this?

How ranking works



Key observation: there is a 1-1 correspondence between strings in L(D) and accepting paths in D

How ranking works



Key observation: there is a 1-1 correspondence between strings in L(D) and accepting paths in D

Step 1: establish an ordering over the alphabet of D

a < b

Step 2: extend this to lexicographical ordering of strings, one symbol at a time

Step 3: order accepting paths based on the lexicographical ordering of the strings that label them

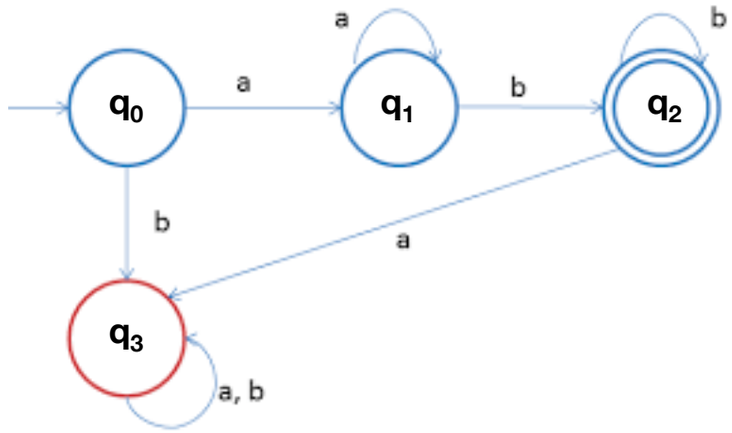
ab < aab < abb < aaab < aabb < …

$$\begin{aligned}\text{rank}(X) = &|\{Y \in L(D) : |Y| = |X| \text{ and } Y < X\}| \\ &+ |Y \in L(D) : |Y| < |X|\}|\end{aligned}$$

How ranking works: Precomputation

algorithm BuildTable(n)

for $q \in Q$ do
  if $q \in F$ then $T[q, 0] \leftarrow 1$
for $i \leftarrow 1, \ldots, n$ do
  for $q \in Q$ do
    for $a \in \Sigma$ do
      $T[q, i] \stackrel{+}{\leftarrow} T[\delta(q, a), i - 1]$



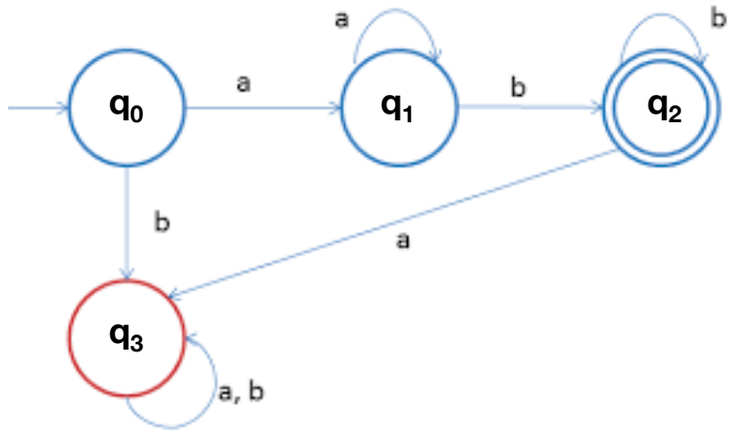
Path length

DFA State	0	1	2	3
q_0				
q_1				
q_2				
q_3				

T[q1,2] = # of accepting paths starting from q1 of length 2

How ranking works: Precomputation

```
algorithm BuildTable(n)
for q ∈ Q do
    if q ∈ F then T[q, 0] ← 1
for i ← 1, . . . , n do
    for q ∈ Q do
        for a ∈ Σ do
            T[q, i] +← T[δ(q, a), i − 1]
```



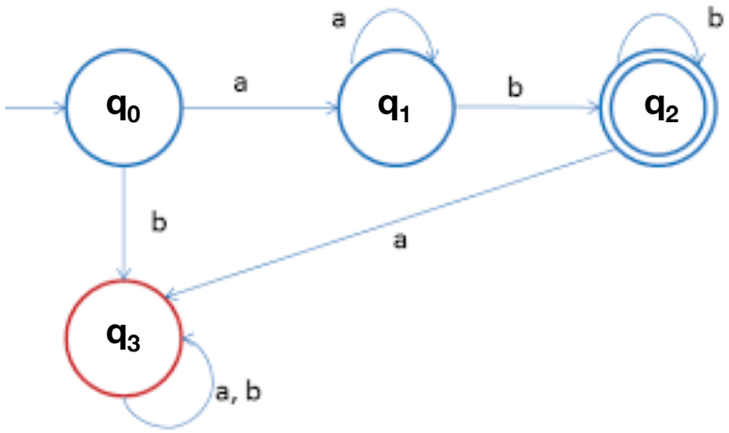
Path length

DFA State	0	1	2	3
q_0	0			
q_1	0			
q_2	1			
q_3	0			

T[q1,2] = # of accepting paths starting from q1 of length 2

How ranking works: Precomputation

```
algorithm BuildTable(n)
for q ∈ Q do
    if q ∈ F then T[q, 0] ← 1
for i ← 1, . . . , n do
    for q ∈ Q do
        for a ∈ Σ do
            T[q, i] +← T[δ(q, a), i − 1]
```

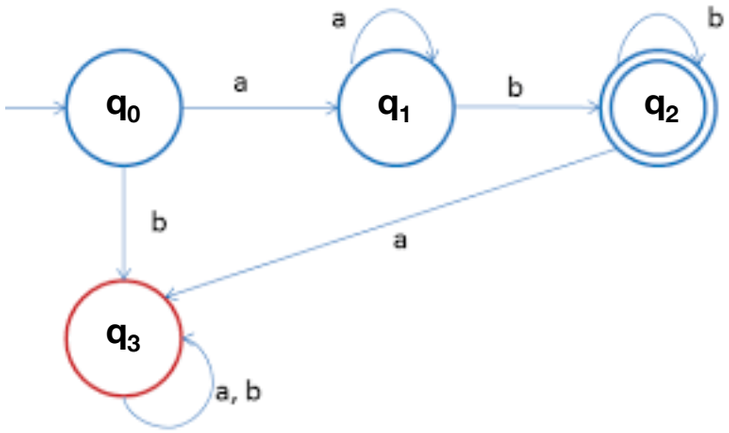


Path length

DFA State	0	1	2	3
q_0	0	0		
q_1	0	1		
q_2	1	1		
q_3	0	0		

How ranking works: Precomputation

```
algorithm BuildTable(n)
for q ∈ Q do
    if q ∈ F then T[q, 0] ← 1
for i ← 1, . . . , n do
    for q ∈ Q do
        for a ∈ Σ do
            T[q, i] +← T[δ(q, a), i − 1]
```



Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

Note: the table is of size (# states x longest string of interest)…

How ranking works: online phase

This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

Begin at the start state, set initial rank to 0

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, X[2]=b, X[3]=b

n=3

$q = q_0$

c = 0

This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

For each position in the string X[1]X[2]X[3]…

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // **X[1]=a**, X[2]=b, X[3]=b
n=3
$q = q_0$
c = 0
i = 1

This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

For each position in the string X[1]X[2]X[3]…

For each symbol < X[i] in the ordering over the symbols

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // **X[1]=a**, X[2]=b, X[3]=b
n=3
$q = q_0$
c = 0
i = 1

This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

For each position in the string X[1]X[2]X[3]…

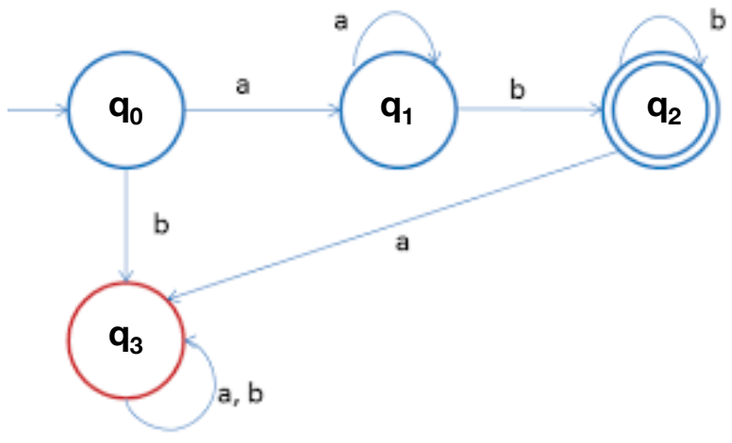
For each symbol < X[i] in the ordering over the symbols

Move forward one step on the path labeled by X

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // **X[1]=a**, X[2]=b, X[3]=b
n=3
q = q_1
c = 0
i = 1



This gives the rank among strings of the same length as X

algorithm $rank(X)$

$q \leftarrow q_0$; $c \leftarrow 0$; $n \leftarrow |X|$
for $i \leftarrow 1, \ldots, n$ do
  for $j \leftarrow 1, \ldots,$ **ord**$(X[i]) - 1$ do
    $c \stackrel{+}{\leftarrow} T[\delta(q, a_j), n - i]$
  $q \leftarrow \delta(q, X[i])$
ret c

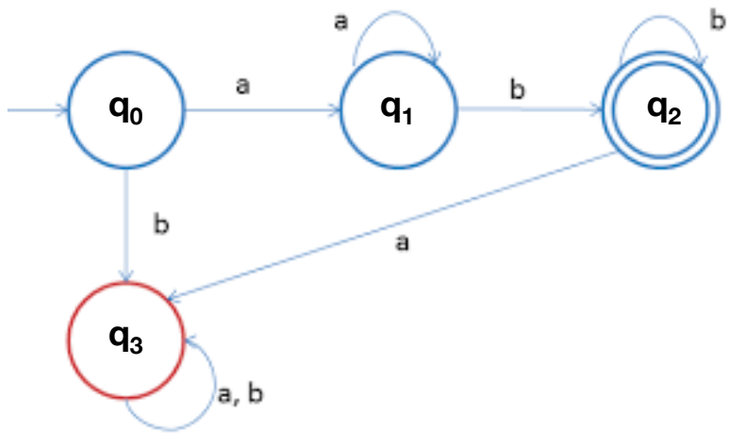
For each position in the string X[1]X[2]X[3]…

For each symbol < X[i] in the ordering over the symbols

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, **X[2]=b**, X[3]=b
n=3
$q = q_1$
c = 0
i = 2



This gives the rank among strings of the same length as X

algorithm $rank(X)$

$q \leftarrow q_0$; $c \leftarrow 0$; $n \leftarrow |X|$
for $i \leftarrow 1, \ldots, n$ do
  for $j \leftarrow 1, \ldots, \mathsf{ord}(X[i]) - 1$ do
    $c \stackrel{+}{\leftarrow} T[\delta(q, a_j), n - i]$
  $q \leftarrow \delta(q, X[i])$
ret c

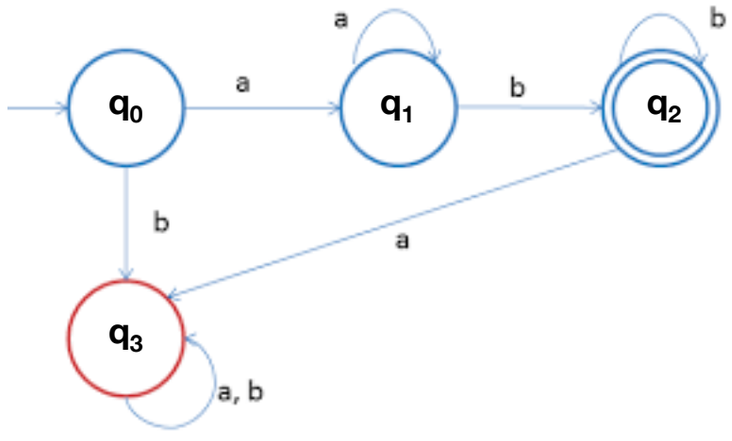
For each position in the string X[1]X[2]X[3]…

For each symbol < X[i] in the ordering over the symbols

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, **X[2]=b**, X[3]=b
n=3
$q = q_1$
c = 0
i = 2



This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

For each position in the string X[1]X[2]X[3]…

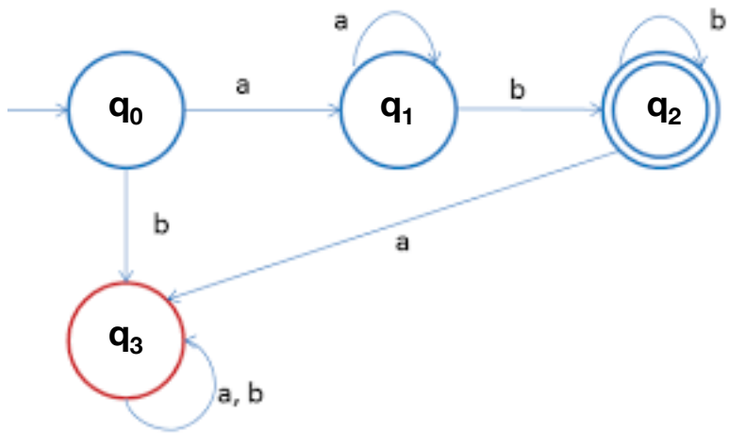
For each symbol < X[i] in the ordering over the symbols

Increase the rank by the # of strings of length n-i that start with the smaller symbol

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, **X[2]=b**, X[3]=b
n=3
q = q_1
c = 1
i = 2



This gives the rank among strings of the same length as X

algorithm $rank(X)$

$q \leftarrow q_0$; $c \leftarrow 0$; $n \leftarrow |X|$
for $i \leftarrow 1, \ldots, n$ do
  for $j \leftarrow 1, \ldots, \mathsf{ord}(X[i]) - 1$ do
    $c \stackrel{+}{\leftarrow} T[\delta(q, a_j), n - i]$
  $q \leftarrow \delta(q, X[i])$
ret c

Move forward

For each position in the string X[1]X[2]X[3]…

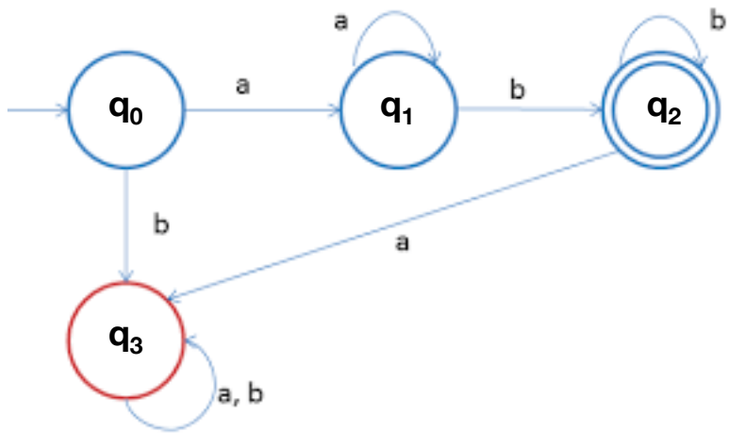
For each symbol < X[i] in the ordering over the symbols

Increase the rank by the # of strings of length n-i that start with the smaller symbol

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, **X[2]=b**, X[3]=b
n=3
q = q_2
c = 1
i = 2



This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
   for j ← 1, . . . , ord(X[i]) − 1 do
      c +← T[δ(q, aj), n − i]
   q ← δ(q, X[i])
ret c
```

For each position in the string X[1]X[2]X[3]…

For each symbol < X[i] in the ordering over the symbols

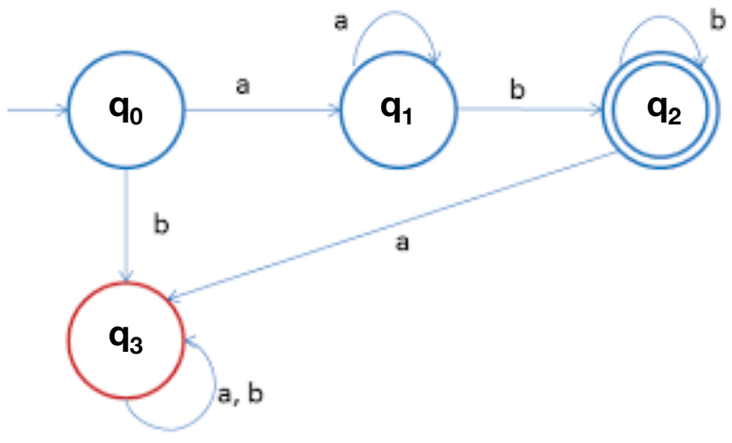
Increase the rank by the # of strings of length n-i that start with the smaller symbol

Move forward

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, X[2]=b, **X[3]=b**
n=3
$q = q_2$
c = 1
i = 3



This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

For each position in the string X[1]X[2]X[3]…

For each symbol < X[i] in the ordering over the symbols

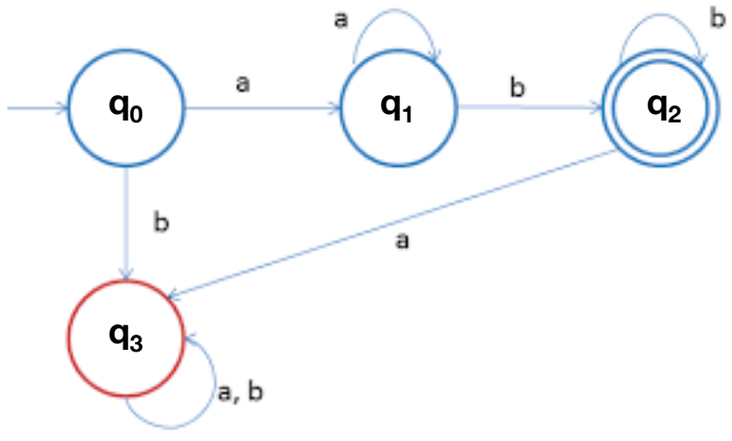
Increase the rank by the # of strings of length n-i that start with the smaller symbol

Move forward

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, X[2]=b, **X[3]=b**
n=3
$q = q_2$
c = 1
i = 3



This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

For each position in the string X[1]X[2]X[3]…

For each symbol < X[i] in the ordering over the symbols

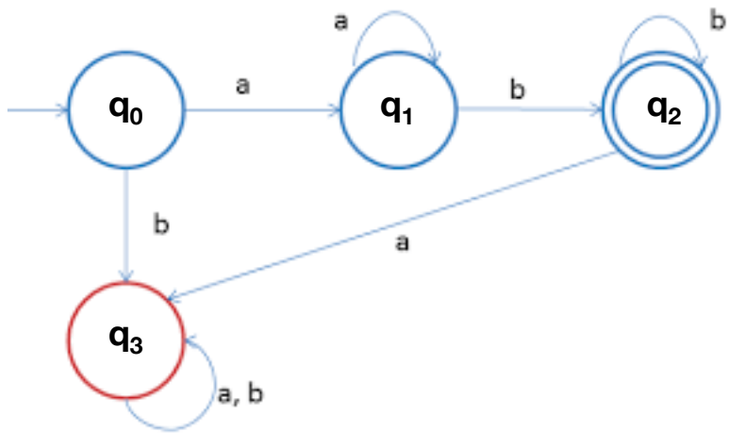
Increase the rank by the # of strings of length n-i that start with the smaller symbol

Move forward

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

rank(abb): // X[1]=a, X[2]=b, **X[3]=b**
n=3
$q = q_2$
c = 1
i = 3



This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

The rank of abb among length-3 strings is 1,
i.e. there is one length-3 string in L(D) that is "less than" abb

This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

Path length

DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

The rank of abb among length-3 strings is 1,
i.e. there is one length-3 string in L(D) that is "less than" abb

… and there is one string in L(D) of length less than 3

This gives the rank among strings of the same length as X

```
algorithm rank(X)
q ← q0 ; c ← 0 ; n ← |X|
for i ← 1, . . . , n do
    for j ← 1, . . . , ord(X[i]) − 1 do
        c +← T[δ(q, aj), n − i]
    q ← δ(q, X[i])
ret c
```

Path length

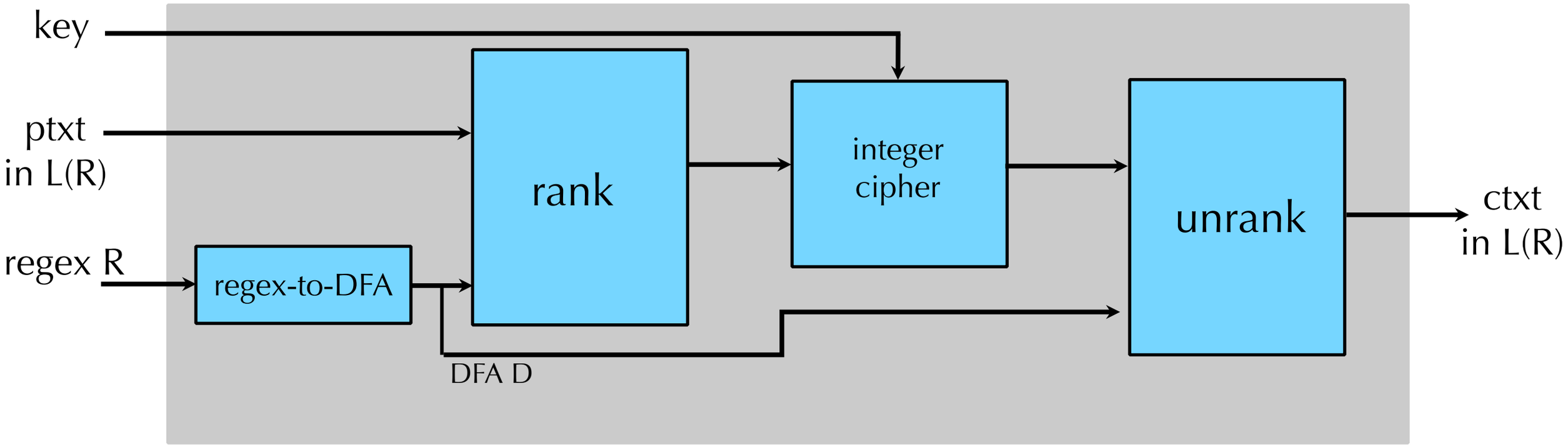
DFA State	0	1	2	3
q_0	0	0	1	2
q_1	0	1	2	3
q_2	1	1	1	1
q_3	0	0	0	0

The rank of abb among length-3 strings is 1,
i.e. there is one length-3 string in L(D) that is "less than" abb

… and there is one string in L(D) of length less than 3

rank(abb) = 2

“rank-encrypt-unrank” FPE construction



But why not rank *directly* from the regex,
instead of converting to an equivalent DFA first?

Come back Thursday.

Ciphers Over "Strange" Domains

Prefix Cipher	n-bit BC ➡ (n-m)-bit BC for **large** m
Cycle-walking Cipher	n-bit BC ➡ (n-m)-bit BC for **small** m
FPE	n-bit BC ➡ cipher over **arbitrary regular language**

Ciphers Over "Strange" Domains

Tom Shrimpton

Florida Institute for Cybersecurity Research
University of Florida

UF Herbert Wertheim College of Engineering
Department of Computer & Information Science & Engineering
UNIVERSITY *of* FLORIDA