Constrained Pseudorandom Functions for Unconstrained Inputs

Apoorvaa Deshpande

Brown University

Joint work with: Venkata Koppula and Brent Waters

TEXAS
The University of Texas at Austin

Pseudorandom Functions

[GGM'84]

Pseudorandom Functions

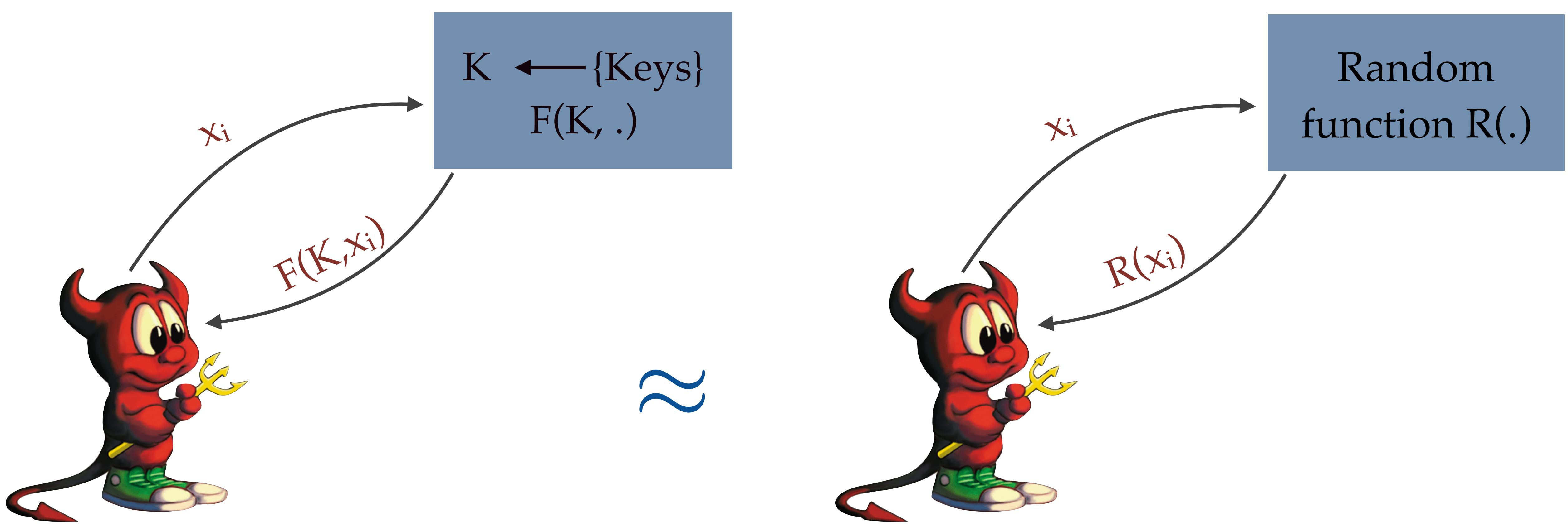
[GGM'84]

F: {Keys} x {Input} ⟶ {Output} is a PRF if it is *indistinguishable* from a truly random function

Pseudorandom Functions

[GGM'84]

F: {Keys} x {Input} ⟶ {Output} is a PRF if it is *indistinguishable* from a truly random function



Pseudorandom Functions

[GGM'84]

Pseudorandom Functions

[GGM'84]

F: {Keys} x {Input} $\longrightarrow$ {Output} is a PRF if it is *like* a truly random function

Pseudorandom Functions

[GGM'84]

F: {Keys} x {Input} → {Output} is a PRF if it is *like* a truly random function

Setup(1^k) → K

Eval(x) → F(K,x)

Pseudorandom Functions

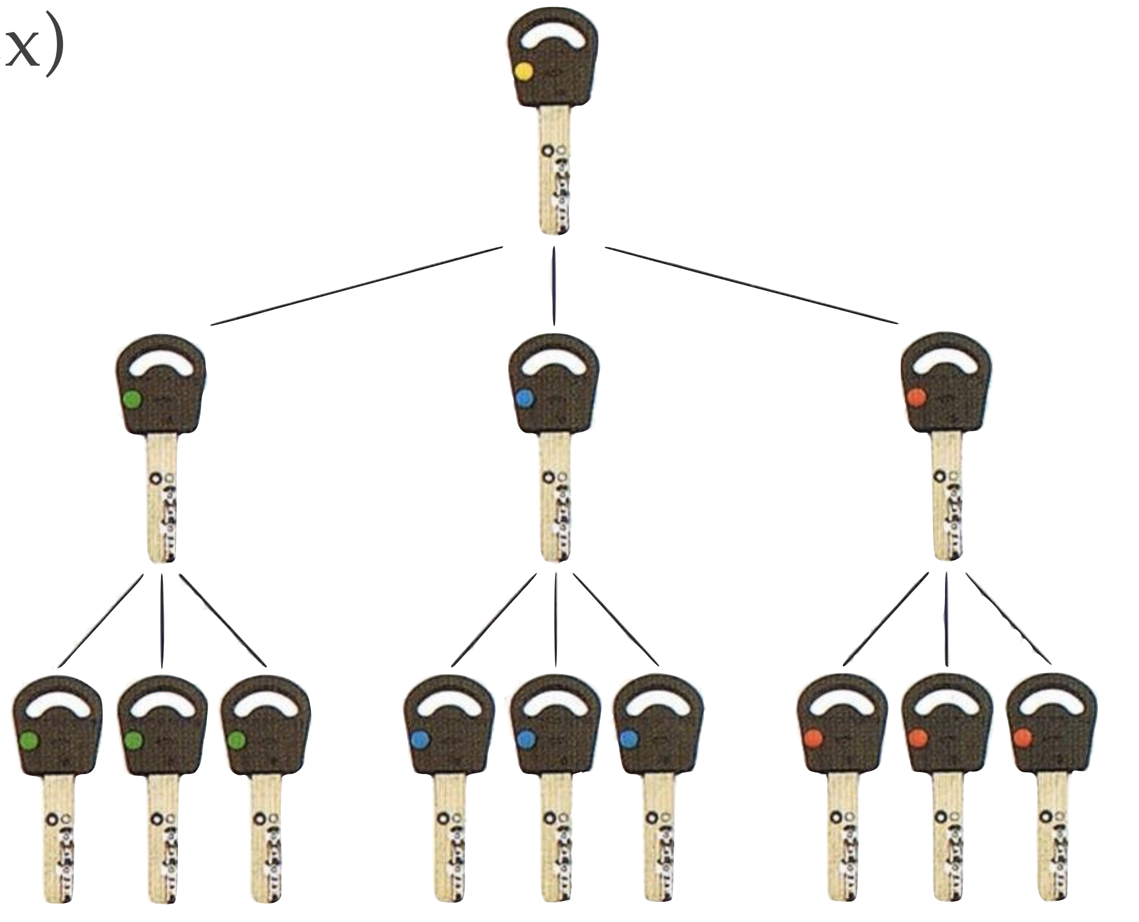
[GGM'84]

F: {Keys} x {Input} ⟶ {Output} is a PRF if it is *like* a truly random function

Setup(1^k) ⟶ K

Eval(x) ⟶ F(K,x)

- What if we don't want to give away our PRF key completely?
- What if we want to give a key that lets us evaluate PRF only on *some* points?



Constrained PRFs

Punctured PRFs: A Type of Constrained PRFs

[SW'14]

Setup → K

Constrain(K,x^*) → K{x^*}

Eval(K{x^*},x) → F(K,x) if x not x^*, fail otherwise

Punctured PRFs: A Type of Constrained PRFs

[SW'14]

Setup	→	K
Constrain(K,x^*)	→	K{x^*}
Eval(K{x^*},x)	→	F(K,x) if x not x^*, fail otherwise

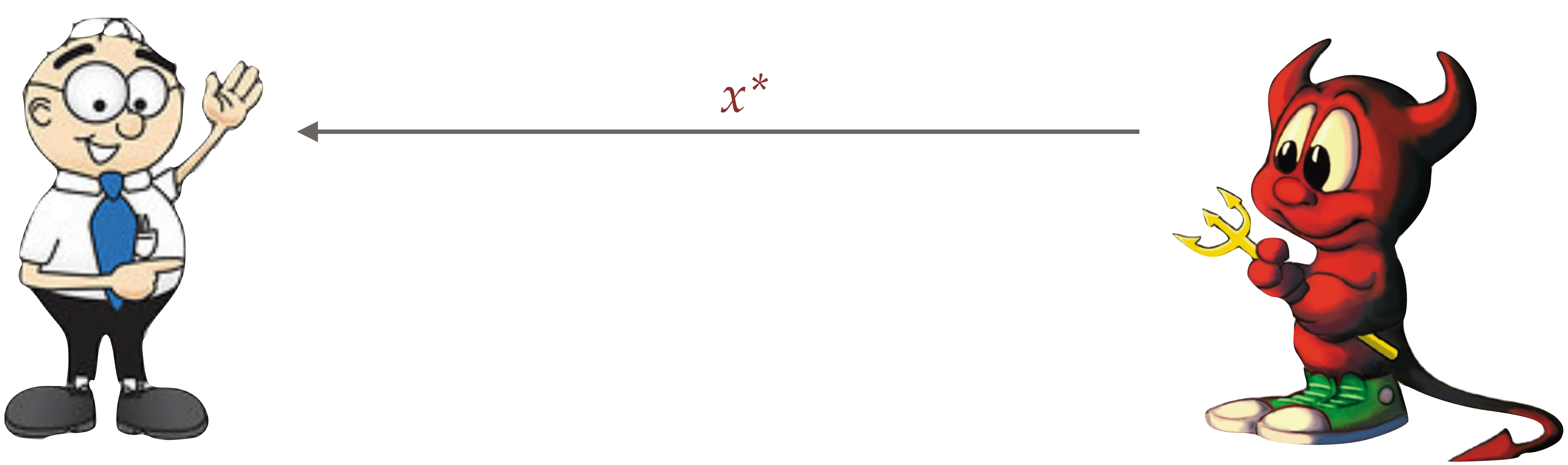
Selective Security

Punctured PRFs: A Type of Constrained PRFs

[SW'14]

Setup	→	K
Constrain(K,x*)	→	K{x*}
Eval(K{x*},x)	→	F(K,x) if x not x*, fail otherwise

Selective Security



<u>Punctured PRFs:</u> A Type of Constrained PRFs

[SW'14]

Setup	→	K
Constrain(K,x*)	→	K{x*}
Eval(K{x*},x)	→	F(K,x) if x not x*, fail otherwise

<u>**Selective Security**</u>

← x^*

b = 0: K{x*}, F(K,x*) →

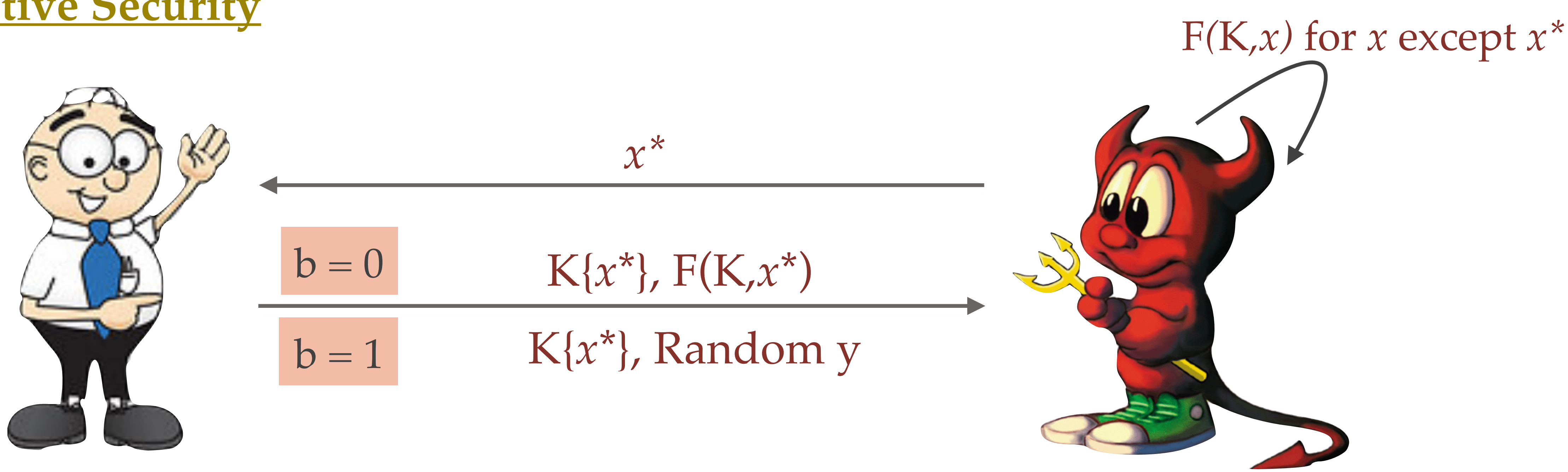
b = 1: K{x*}, Random y

Punctured PRFs: A Type of Constrained PRFs

[SW'14]

Setup	→	K
Constrain(K,x^*)	→	K{x^*}
Eval(K{x^*},x)	→	F(K,x) if x not x^*, fail otherwise

Selective Security

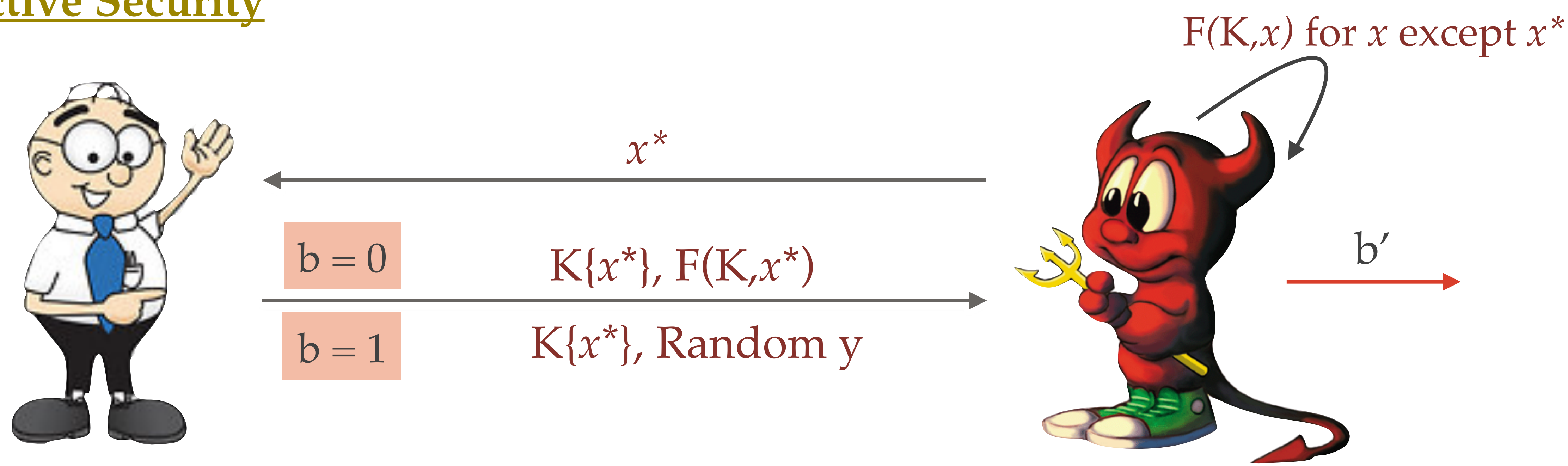


Punctured PRFs: A Type of Constrained PRFs

[SW'14]

Setup →	K
Constrain(K,x^*) →	K{x^*}
Eval(K{x^*},x) →	F(K,x) if x not x^*, fail otherwise

Selective Security



Punctured PRFs: A Type of Constrained PRFs

[SW'14]

Setup	→	K
Constrain(K,x^*)	→	K{x^*}
Eval(K{x^*},x)	→	F(K,x) if x not x^*, fail otherwise

Selective Security

x^* ←

b = 0: K{x^*}, F(K,x^*) →

b = 1: K{x^*}, Random y →

F(K,x) for x except x^*

b′ →

Pr[$\mathcal{A}$ wins] = Pr[b = b′] = 1/2 + negligible

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Setup	→ K
Constrain(K,*C*)	→ K{C}
Eval(K{C},*x*)	→ F(K,*x*) if *x* satisfies constraint C, fail otherwise

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Setup	→	K
Constrain(K,*C*)	→	K{C}
Eval(K{C},*x*)	→	F(K,*x*) if *x* satisfies constraint C, fail otherwise

Selective Security:

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Setup	⟶	K
Constrain(K,*C*)	⟶	K{C}
Eval(K{C},x)	⟶	F(K,x) if x satisfies constraint C, fail otherwise

Selective Security:

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Setup → K

Constrain(K,C) → K{C}

Eval(K{C},x) → F(K,x) if x satisfies constraint C, fail otherwise

Selective Security:

x^* ←

b = 0 y = F(K,x^*) →

b = 1 Truly random y

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Setup → K

Constrain(K,*C*) → K{C}

Eval(K{C},*x*) → F(K,x) if x satisfies constraint C, fail otherwise

Selective Security:

x^* ←

b = 0: y = F(K,x^*) →

b = 1: Truly random y

C s.t. C(x^*) = 0 ←

K{C} →

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Setup → K

Constrain(K,*C*) → K{C}

Eval(K{C},*x*) → F(K,*x*) if *x* satisfies constraint C, fail otherwise

Selective Security:

← x^*

b = 0 y = F(K,x^*) →

b = 1 Truly random y

← C s.t. C(x^*) = 0

K{C} →

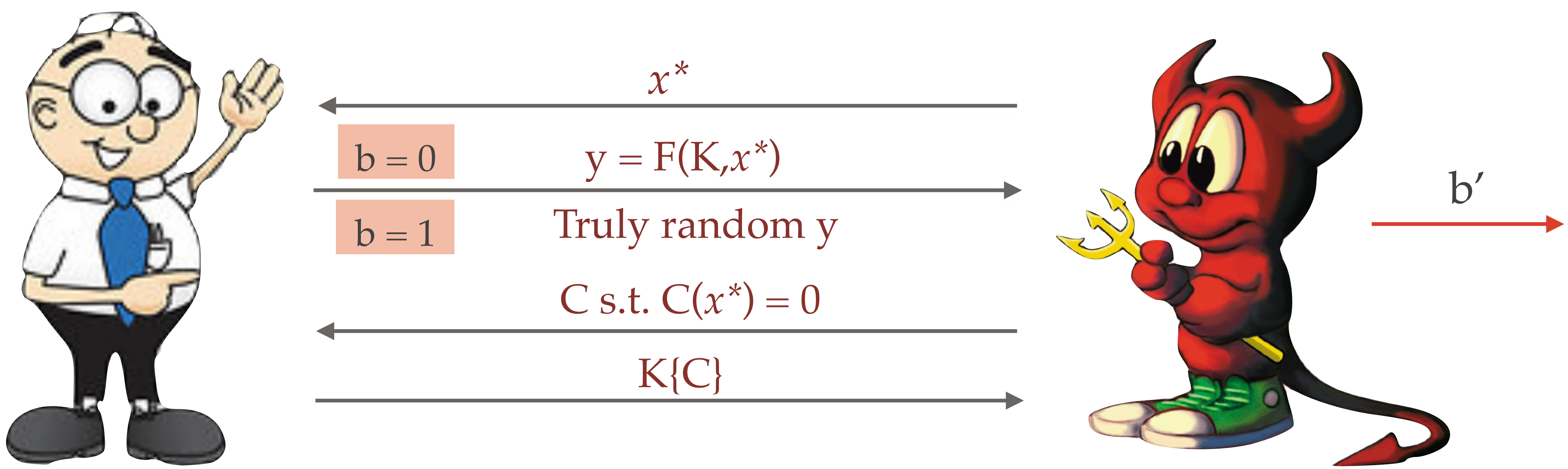
b′ →

Constrained PRFs

[BW'13], [BGI'14], [KPTZ'13]

Setup	→ K
Constrain(K,C)	→ K{C}
Eval(K{C},x)	→ F(K,x) if x satisfies constraint C, fail otherwise

Selective Security:



Pr[$\mathcal{A}$ wins] = Pr[b = b′] = 1/2 + negligible

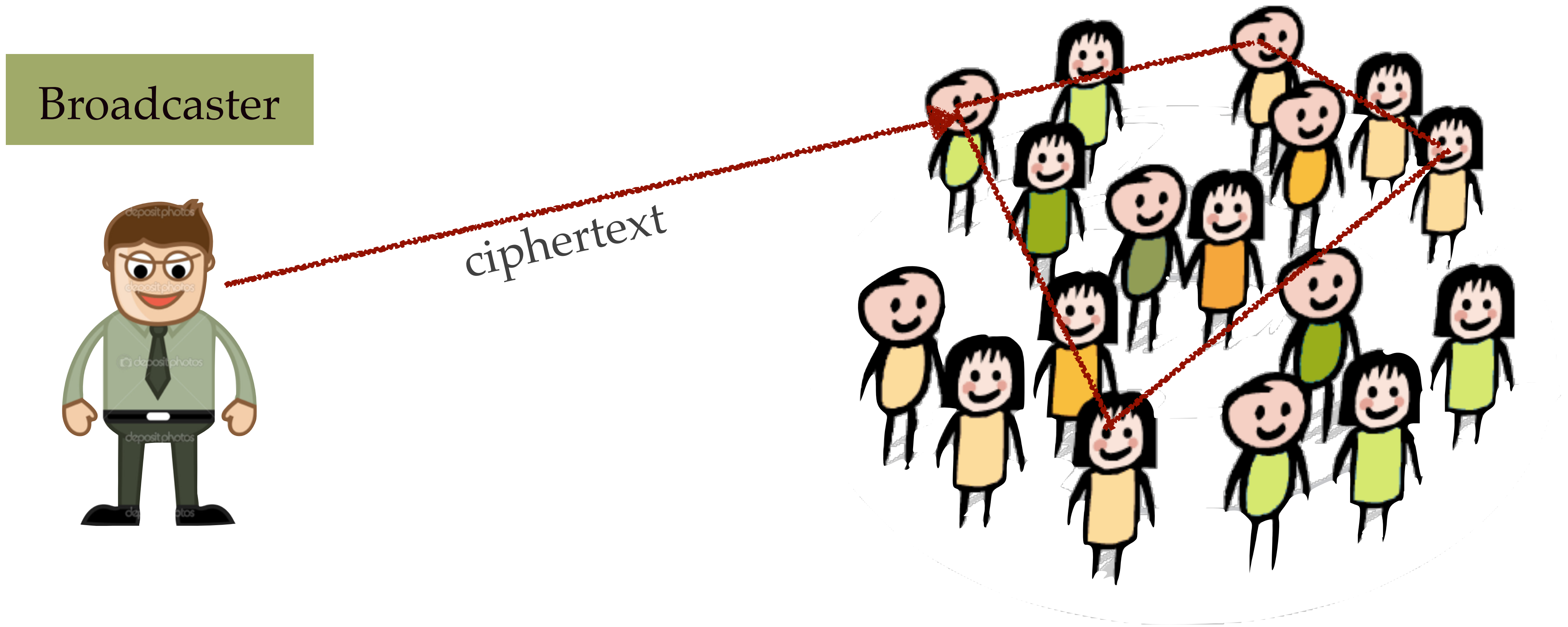
Motivating Example

Motivating Example

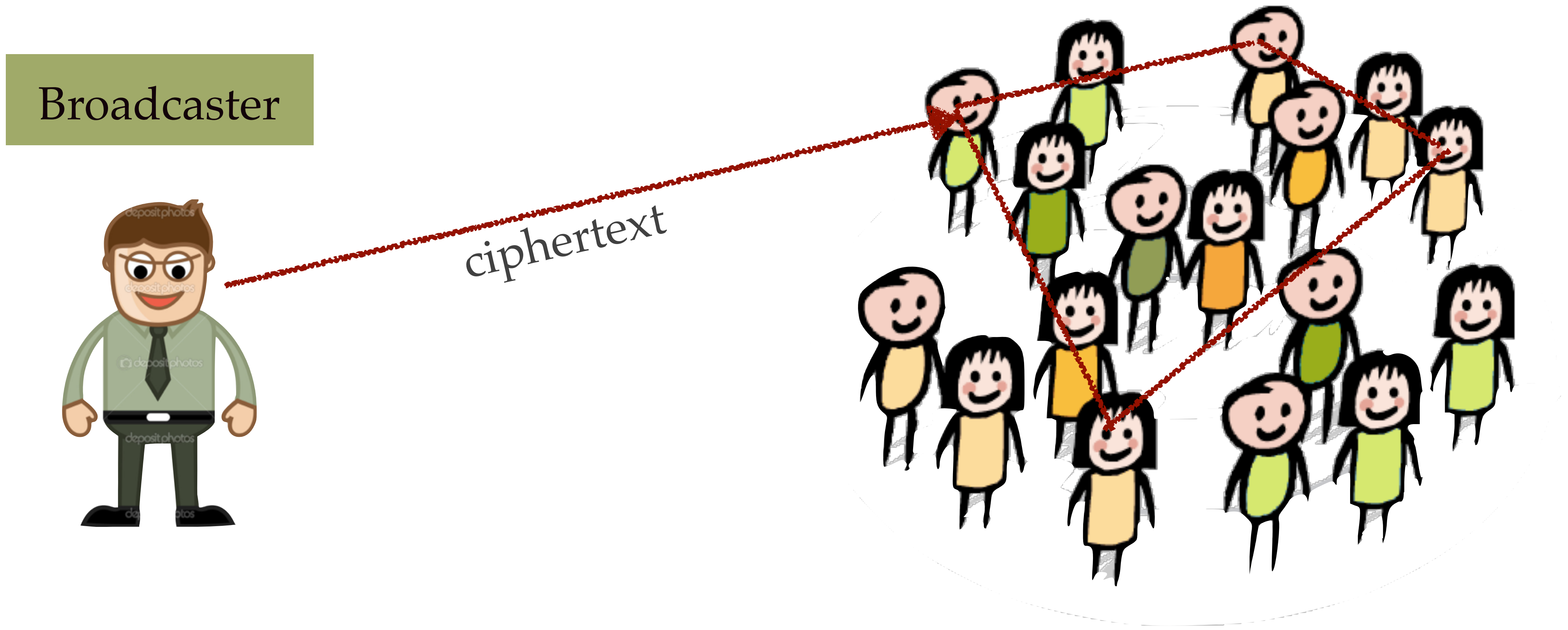
Motivating Example

Broadcaster

Motivating Example

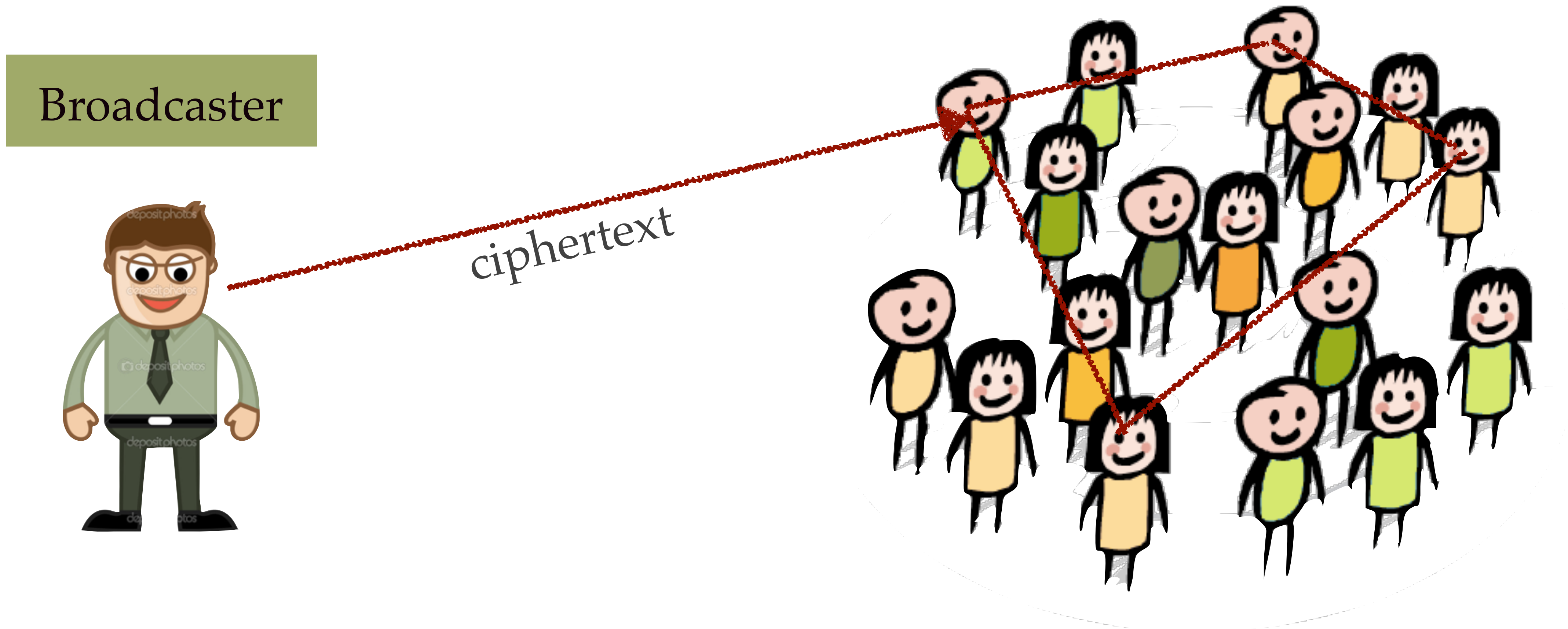


Motivating Example



Broadcast Encryption [FN'93]

Motivating Example



Broadcast Encryption [FN'93]

- Lets you encrypt messages for a specific subset
- No one outside the subset can learn the message
- Can be constructed from constrained PRFs (bit-fixing PRFs)

[BW'13], [BWZ'14]

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

GGM as a Constrained PRF

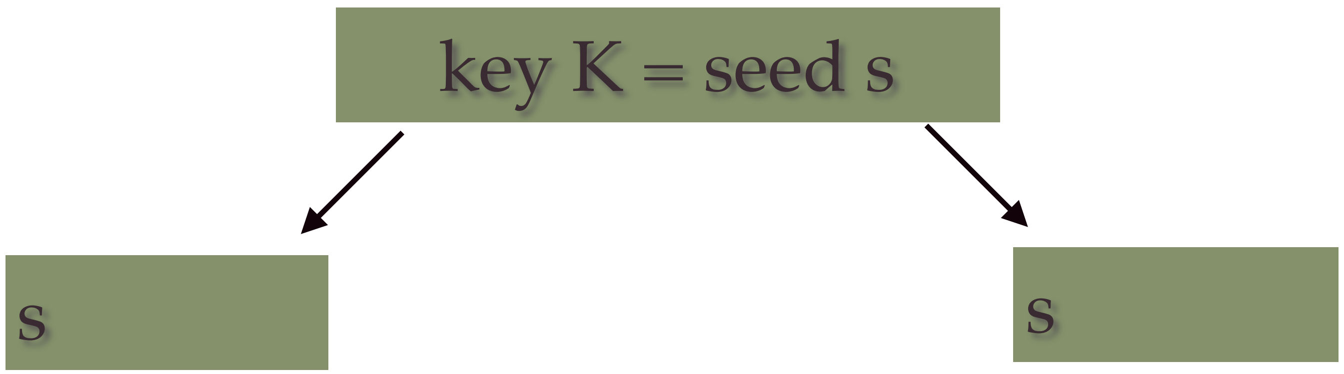
[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \ldots x_n) = G_{x_n}(\ldots G_{x_2}(G_{x_1}(s)) \ldots)$$

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

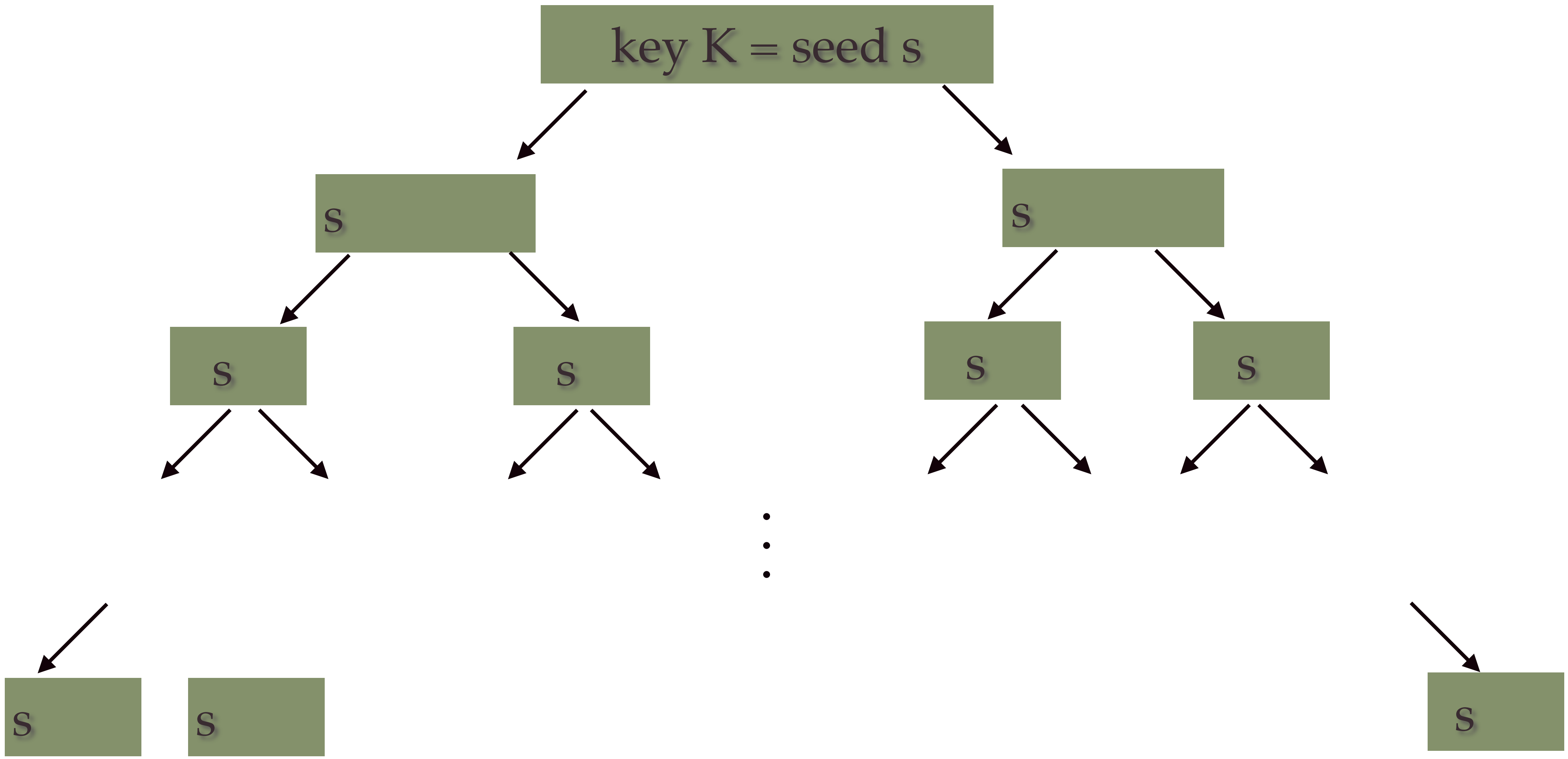
$$F(s, x = x_1 x_2 \ldots x_n) = G_{x_n}(\ldots G_{x_2}(G_{x_1}(s)) \ldots)$$



GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

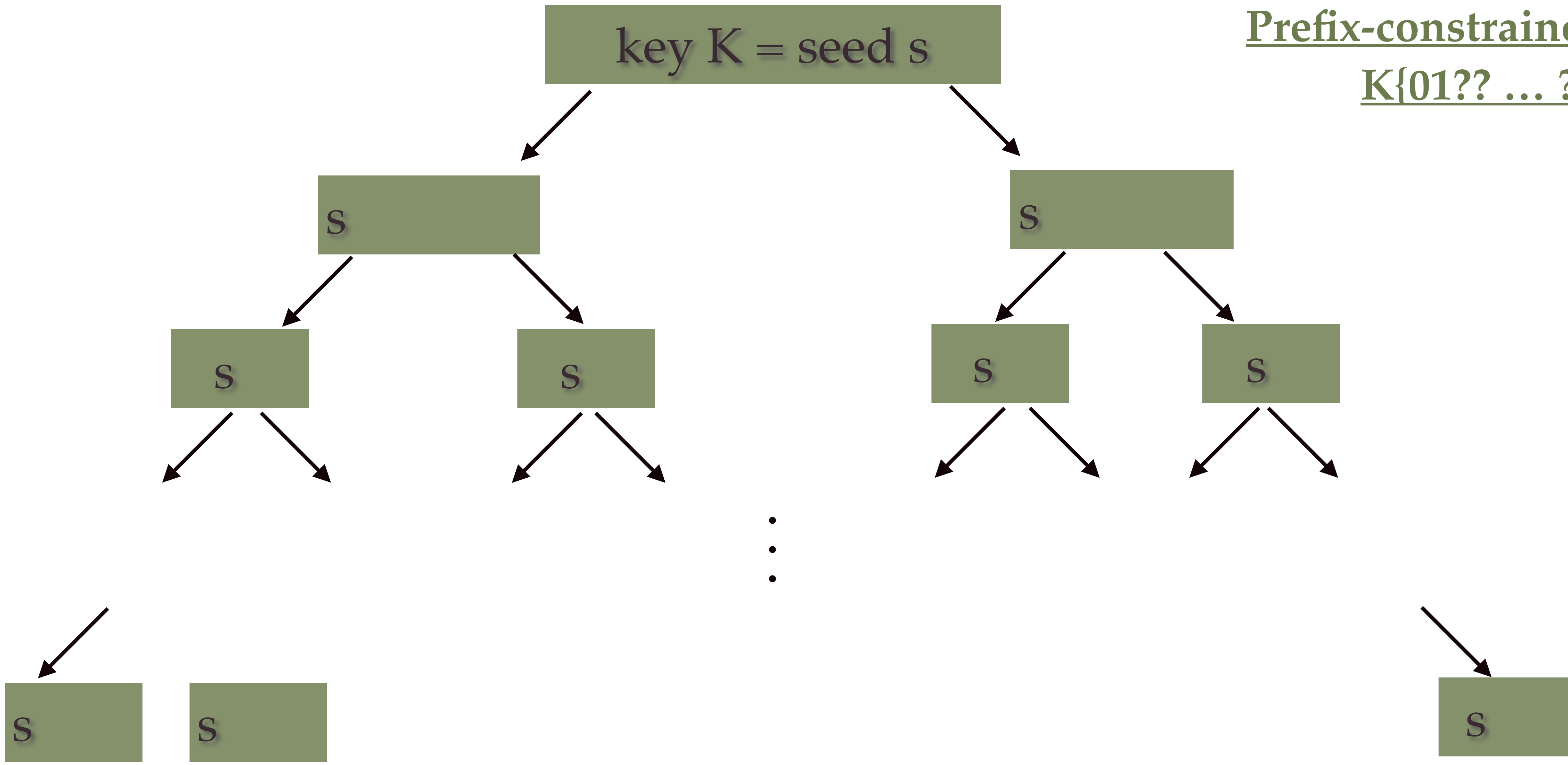
$$F(s, x = x_1 x_2 \ldots x_n) = G_{x_n}(\ldots G_{x_2}(G_{x_1}(s)) \ldots)$$



GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \dots x_n) = G_{x_n}(\dots G_{x_2}(G_{x_1}(s)) \dots)$$

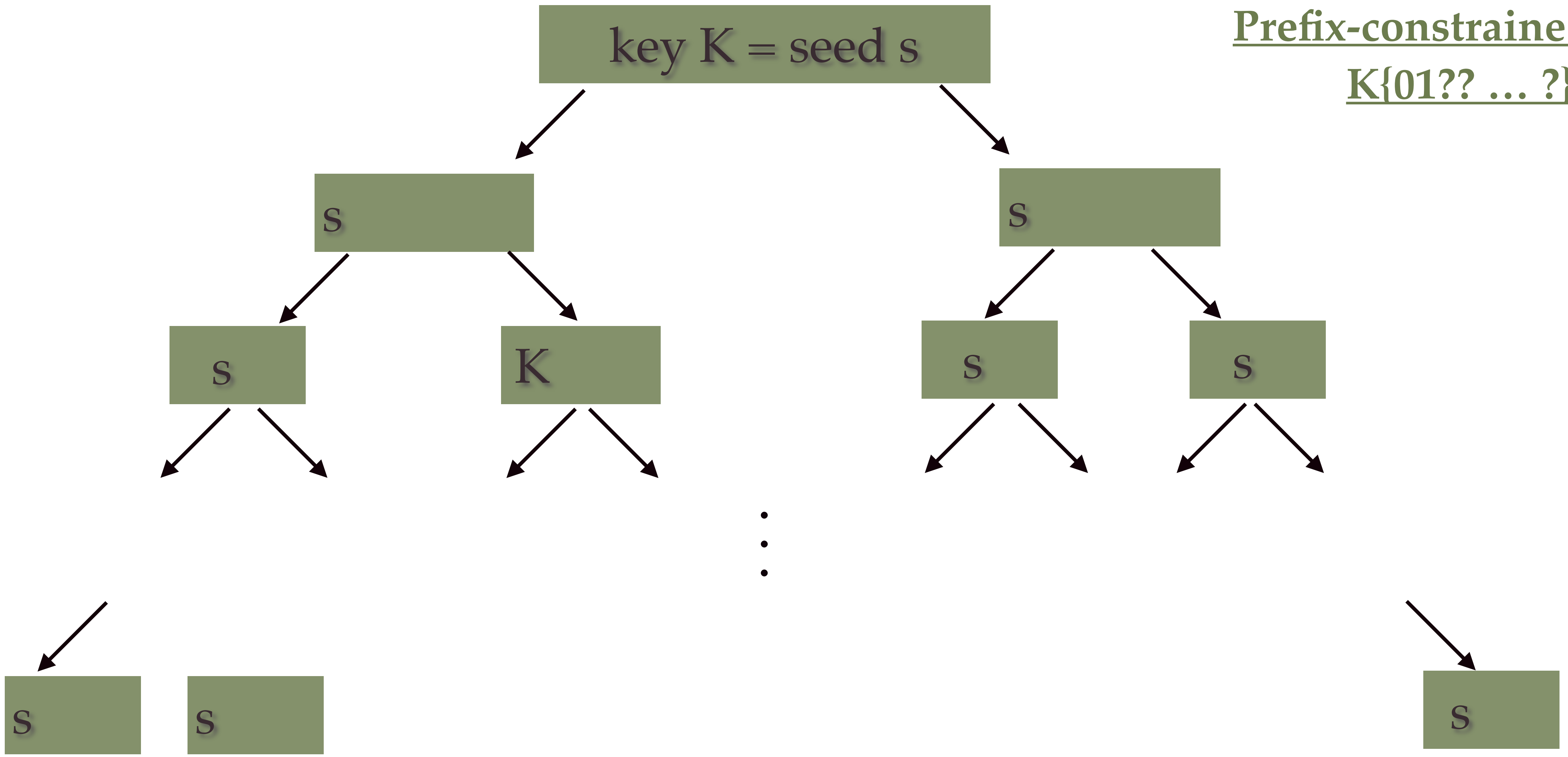


Prefix-constrained key:
K{01?? … ?}

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \ldots x_n) = G_{x_n}(\ldots G_{x_2}(G_{x_1}(s)) \ldots)$$

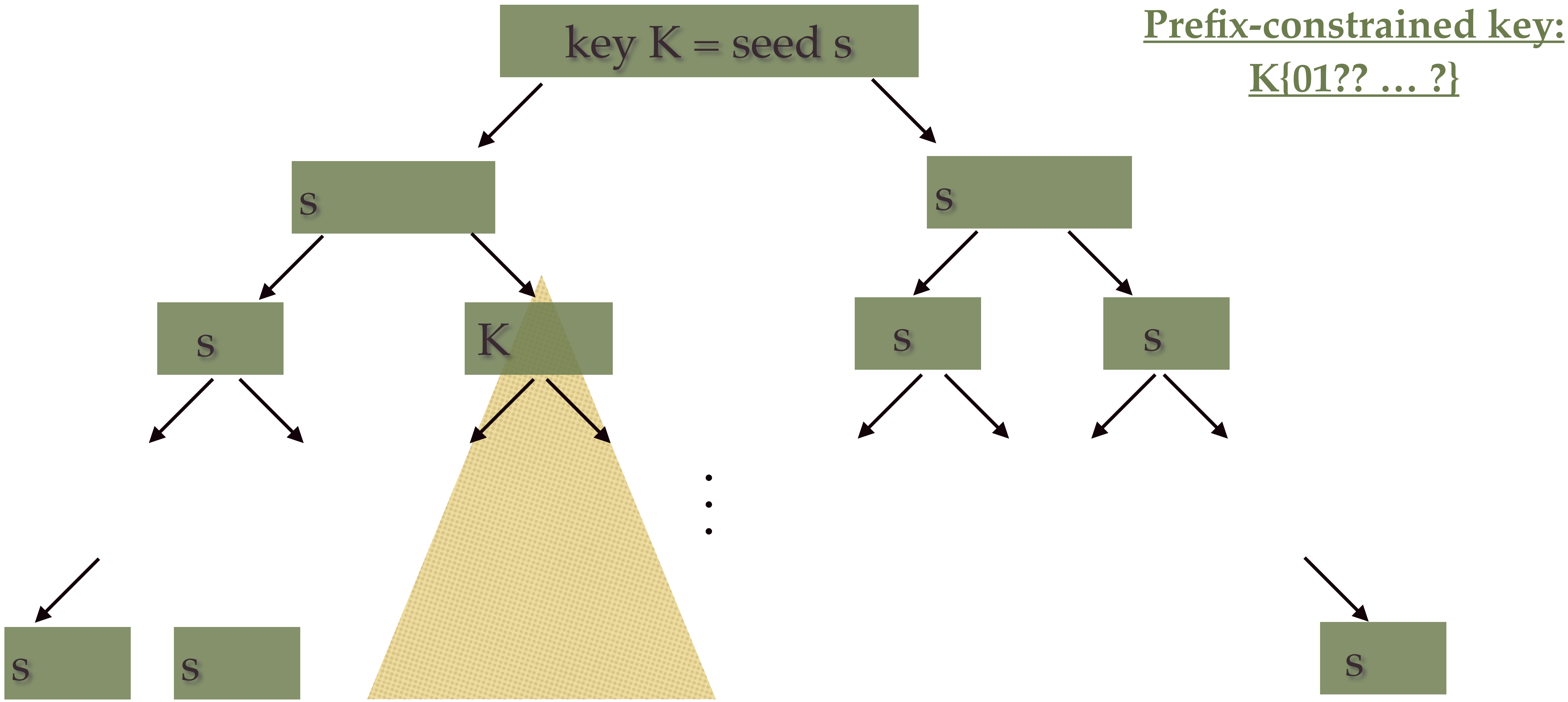


Prefix-constrained key:
K{01?? … ?}

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

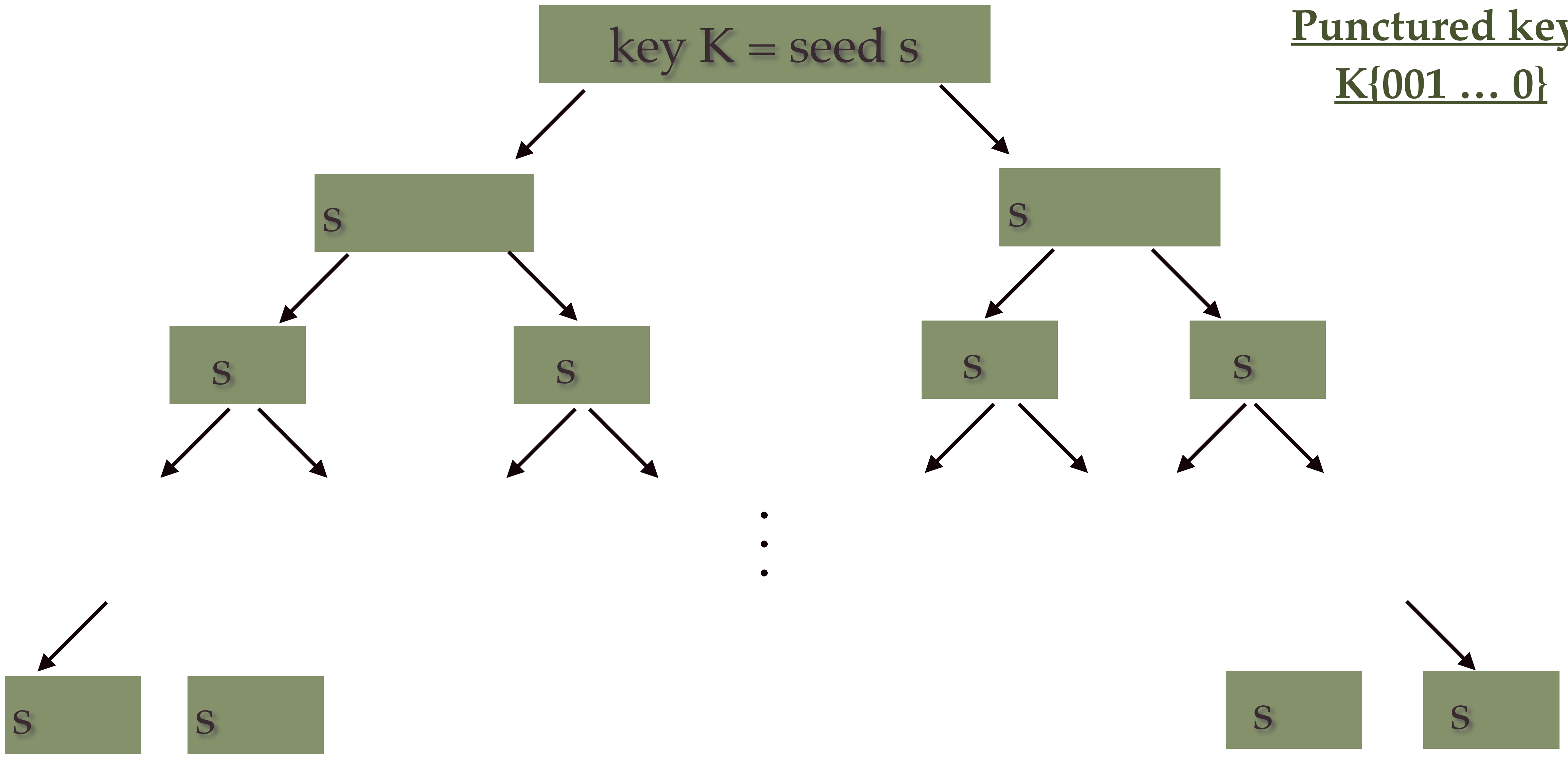
$$F(s, x = x_1 x_2 \ldots x_n) = G_{x_n}(\ldots G_{x_2}(G_{x_1}(s)) \ldots)$$



GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1x_2 \dots x_n) = G_{x_n}(\dots G_{x_2}(G_{x_1}(s)) \dots)$$

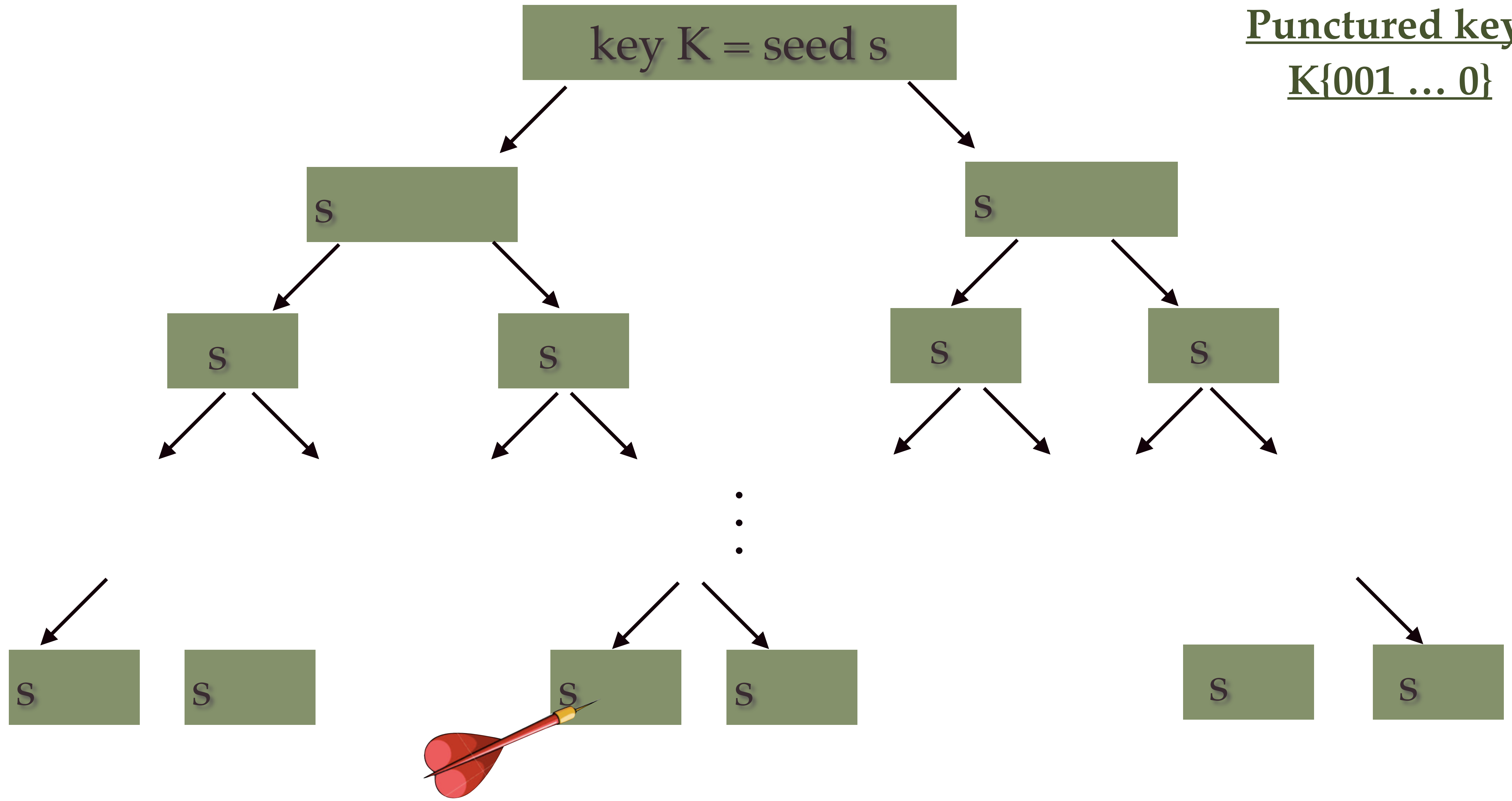


Punctured key:
K{001 … 0}

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \dots x_n) = G_{x_n}(\dots G_{x_2}(G_{x_1}(s)) \dots)$$



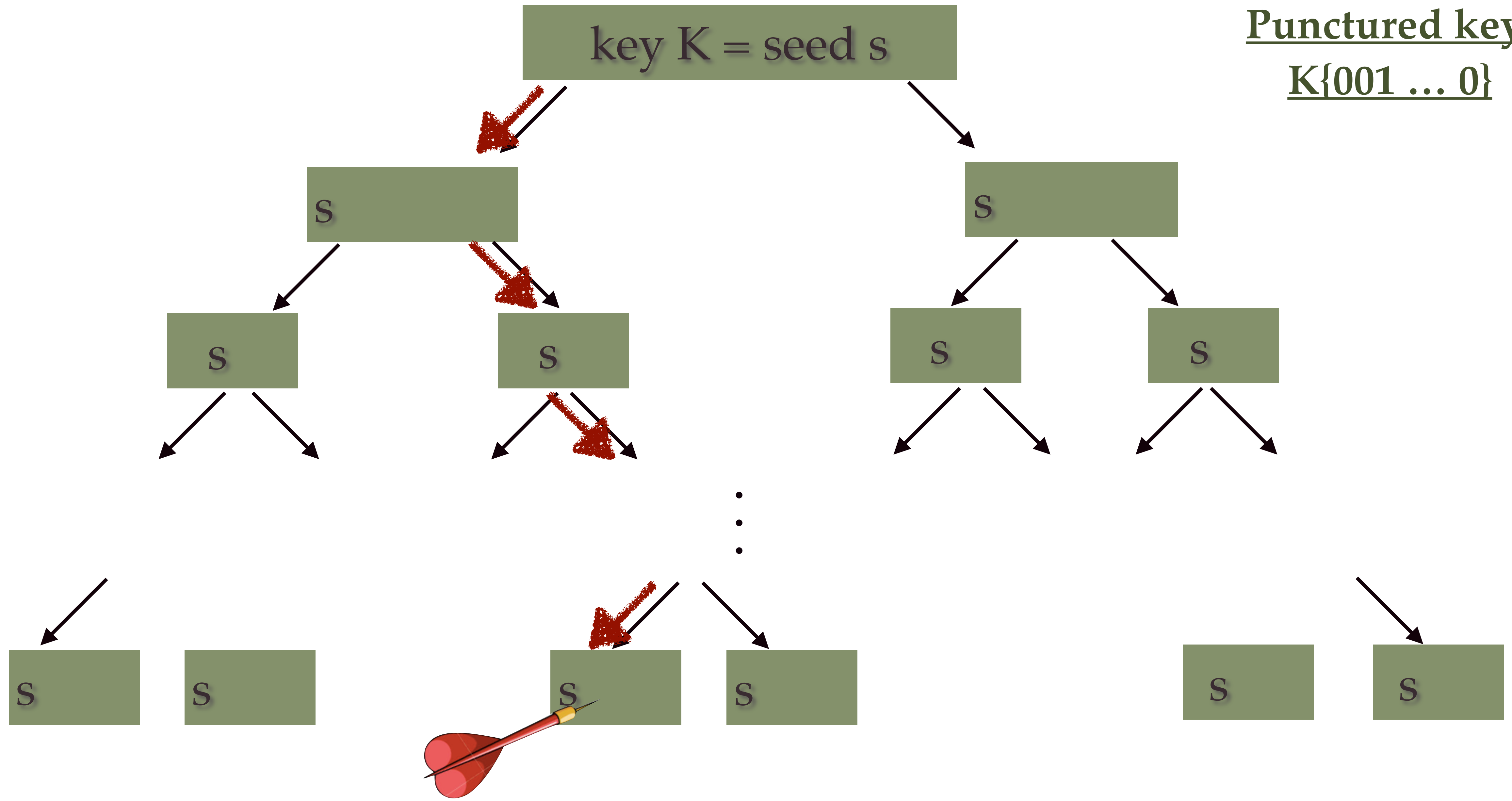
Punctured key:
K{001 … 0}

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \dots x_n) = G_{x_n}(\dots G_{x_2}(G_{x_1}(s)) \dots)$$

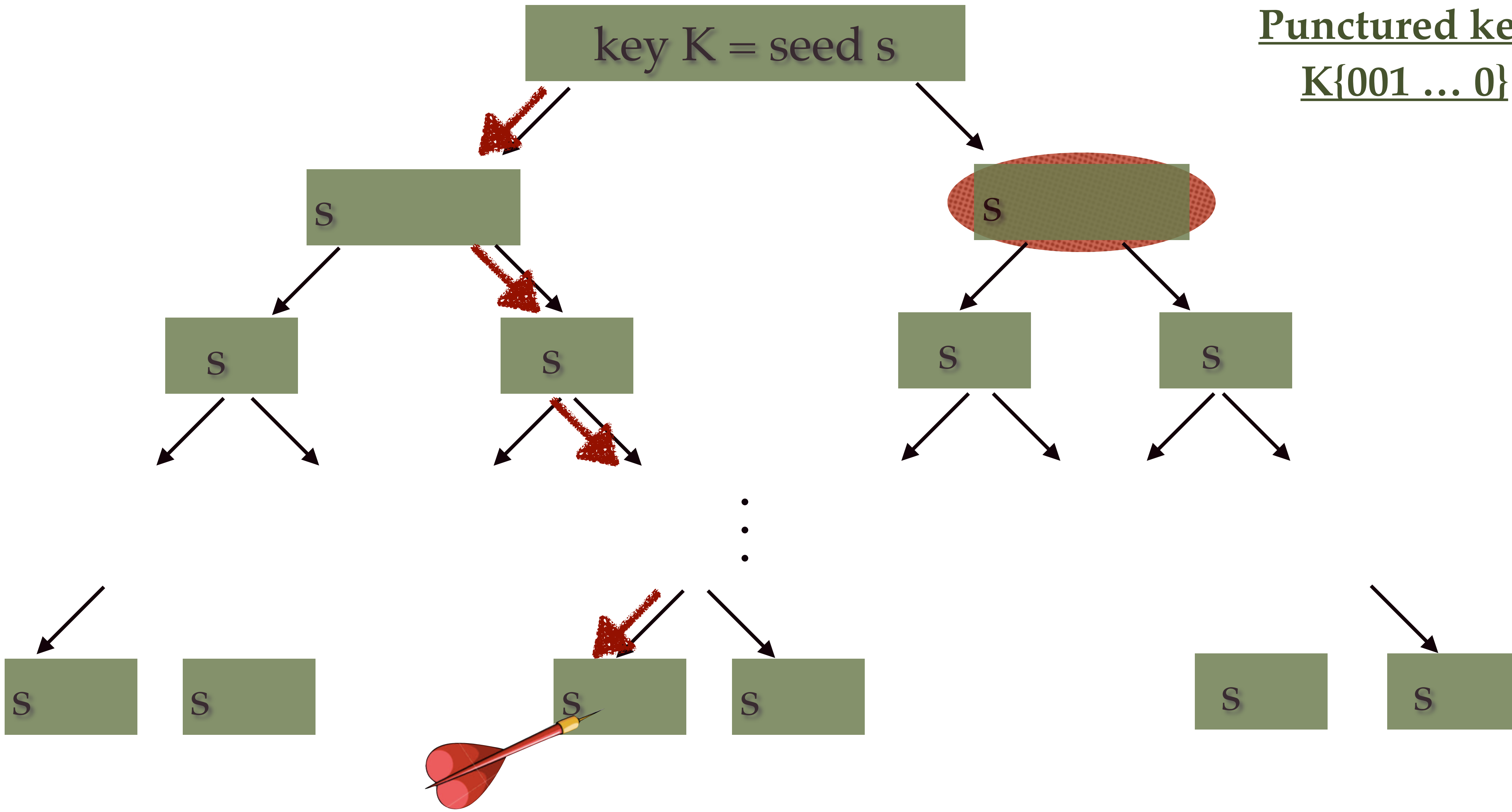
Punctured key:
K{001 … 0}



GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \dots x_n) = G_{x_n}(\dots G_{x_2}(G_{x_1}(s))\dots)$$

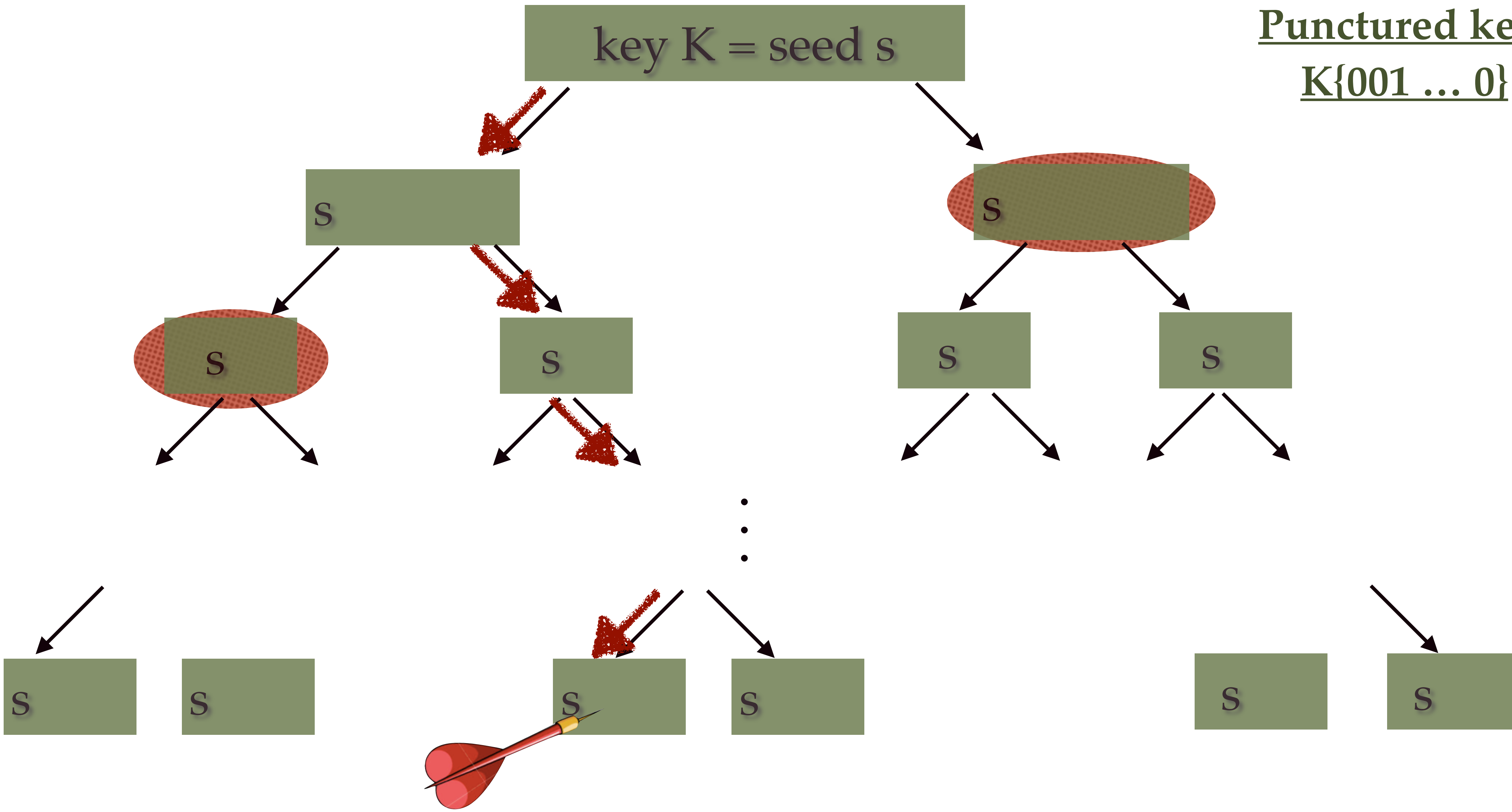


Punctured key:
K{001 … 0}

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \dots x_n) = G_{x_n}(\dots G_{x_2}(G_{x_1}(s)) \dots)$$

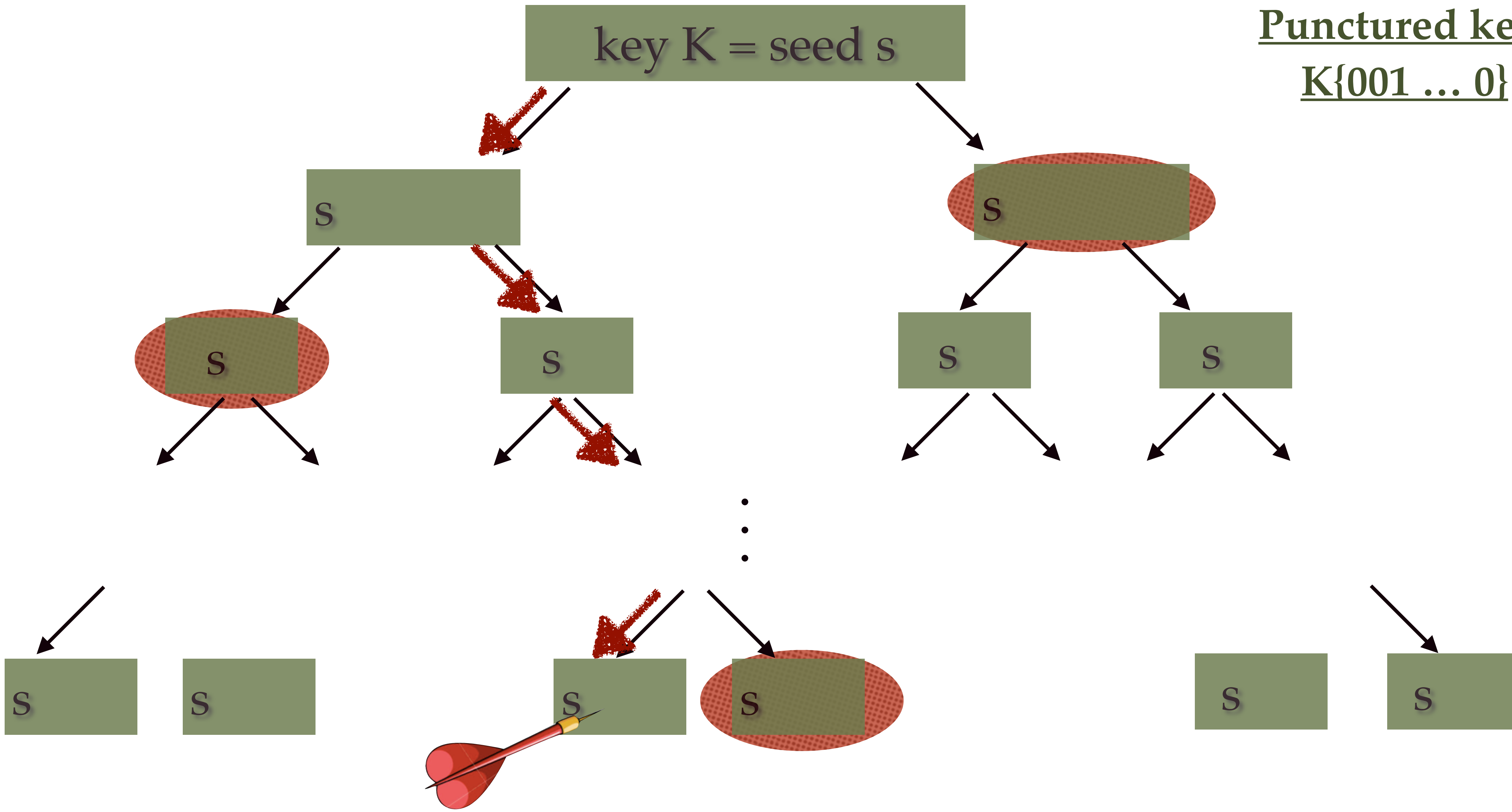


Punctured key:
K{001 … 0}

GGM as a Constrained PRF

[BW'13], [BGI'14], [KPTZ'13]

$$F(s, x = x_1 x_2 \dots x_n) = G_{x_n}(\dots G_{x_2}(G_{x_1}(s))\dots)$$



Punctured key:
K{001 … 0}

CPRF Generically

- Generic CPRF: Constraint expressed as an arbitrary circuit
- Known only from strong cryptographic assumptions

CPRF Generically

- Generic CPRF: Constraint expressed as an arbitrary circuit
- Known only from strong cryptographic assumptions

Possible Construction

CPRF Generically

- <u>Generic CPRF:</u> Constraint expressed as an arbitrary circuit
- Known only from strong cryptographic assumptions

Possible Construction

x

↓

If C(x) = 1

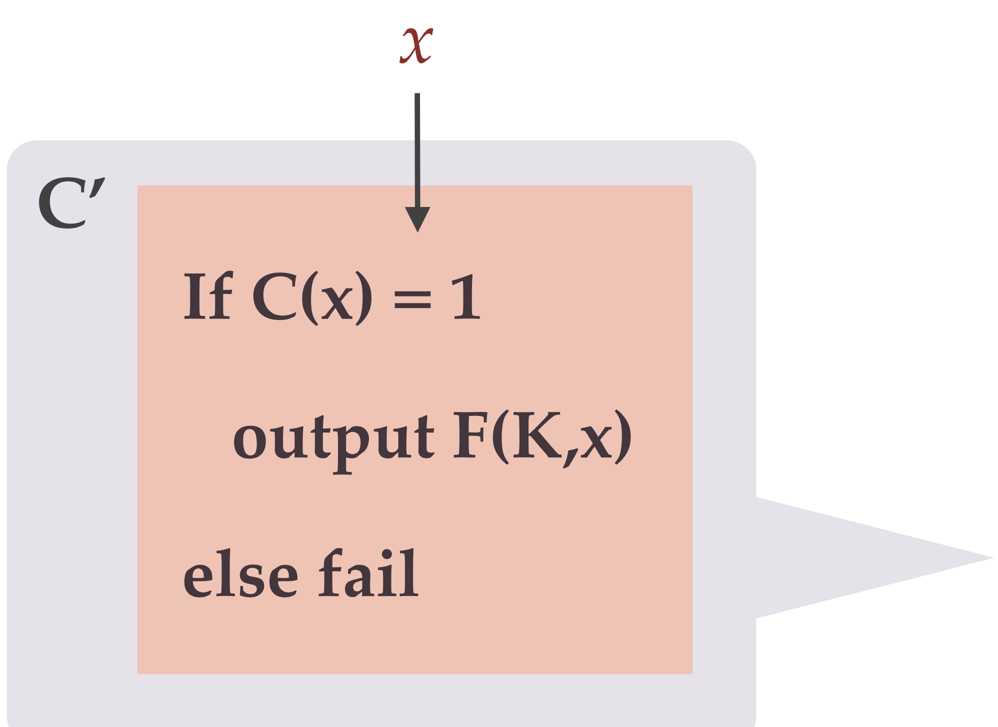
output F(K,x)

else fail

CPRF Generically

- <u>Generic CPRF:</u> Constraint expressed as an arbitrary circuit
- Known only from strong cryptographic assumptions

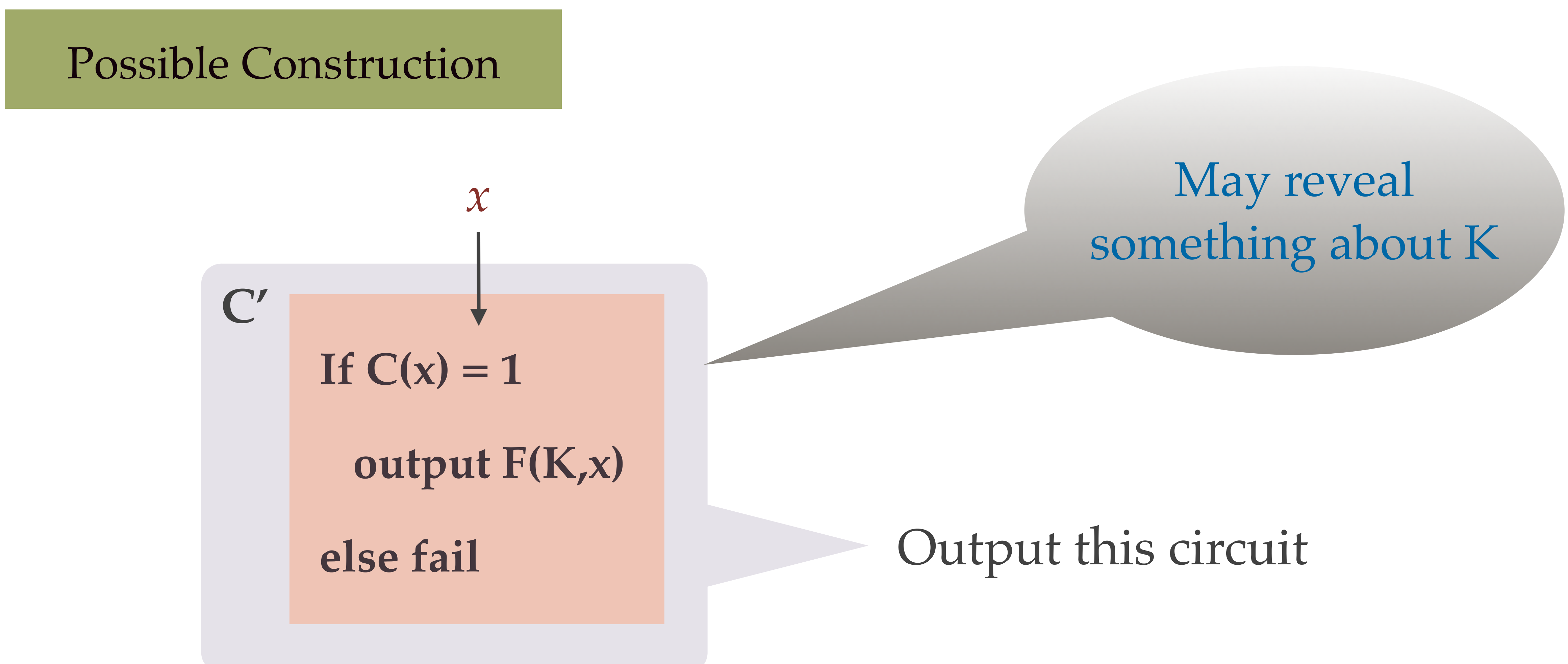
Possible Construction



Output this circuit

CPRF Generically

- Generic CPRF: Constraint expressed as an arbitrary circuit
- Known only from strong cryptographic assumptions



Program Obfuscation

Program Obfuscation

Program Obfuscation

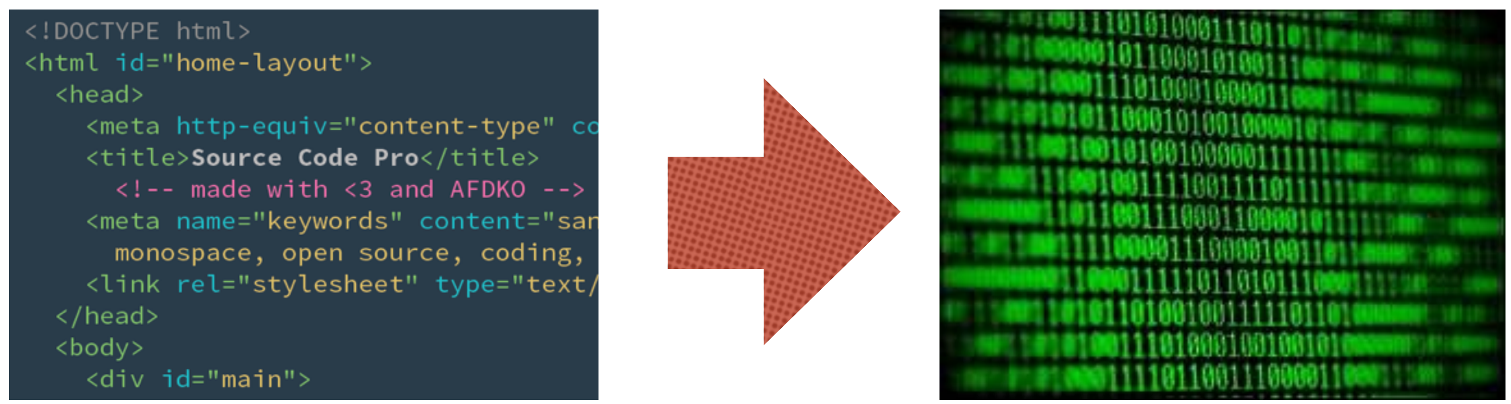
Program Obfuscation

Obfuscated Program

Program Obfuscation

Program Obfuscation

[BGI+’01]



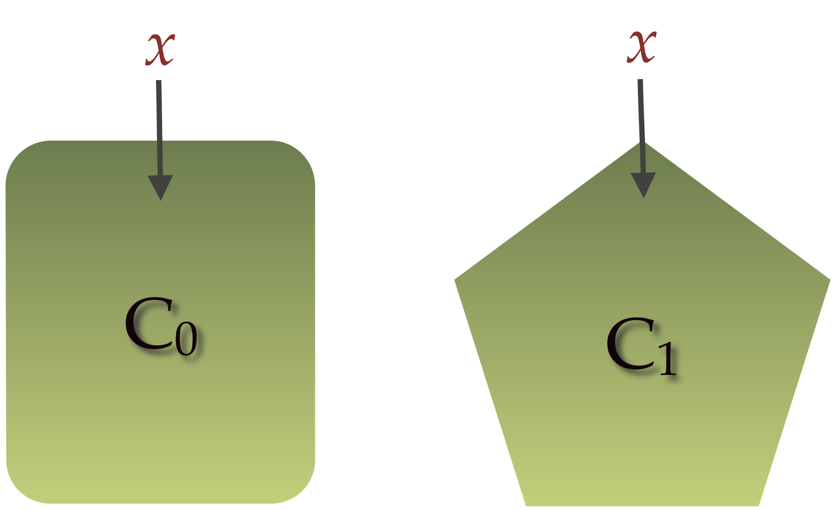
Obfuscated Program

Program Obfuscation

Program Obfuscation [BGI+'01]

Obfuscated Program

Indistinguishability Obfuscation



$C_0(x) = C_1(x)$ for all x

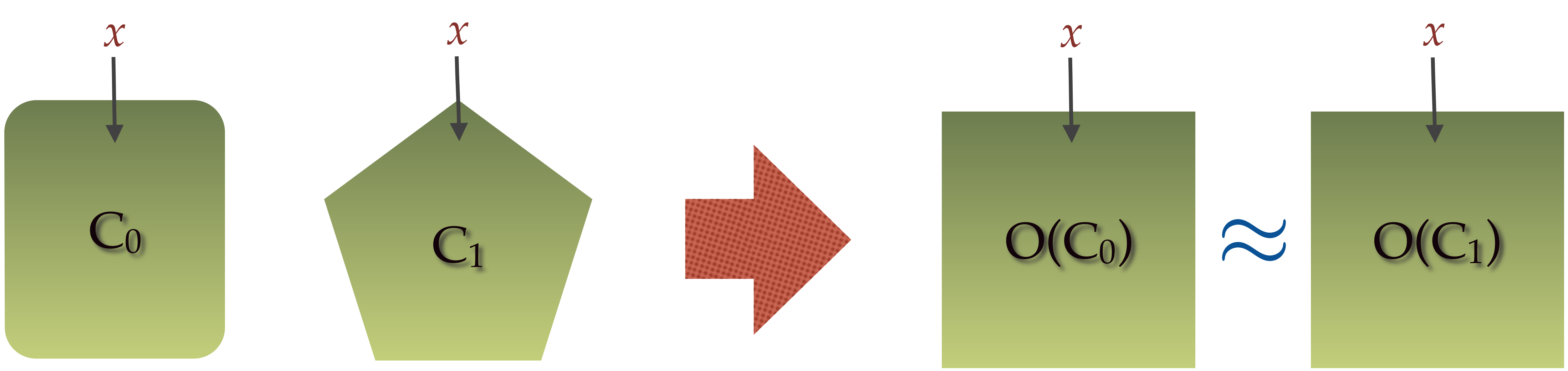
Program Obfuscation

Program Obfuscation

[BGI+'01]

Obfuscated Program

Indistinguishability Obfuscation



$C_0(x) = C_1(x)$ for all x

Program Obfuscation

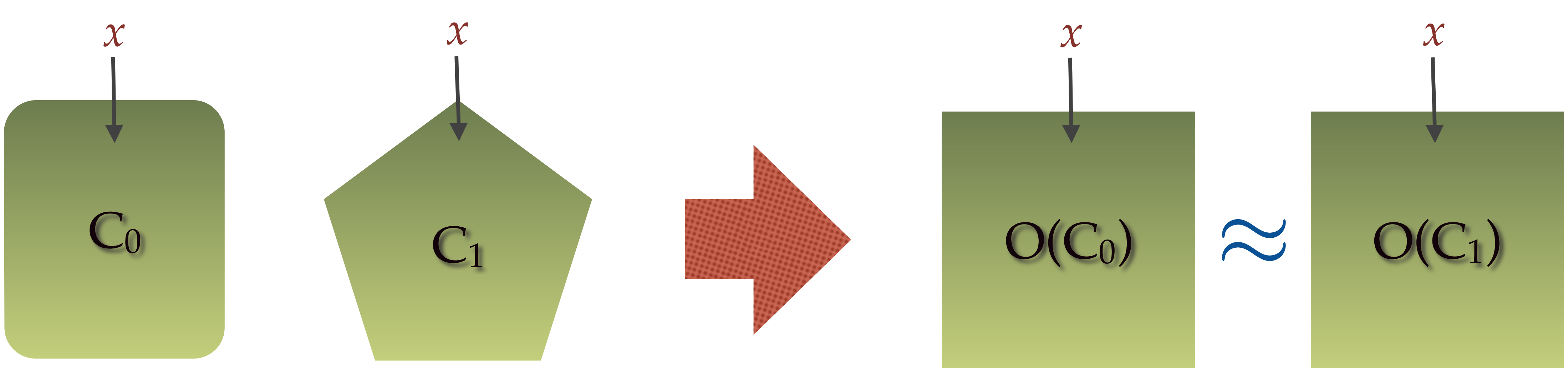
Program Obfuscation

[BGI+'01]

Obfuscated Program

Indistinguishability Obfuscation

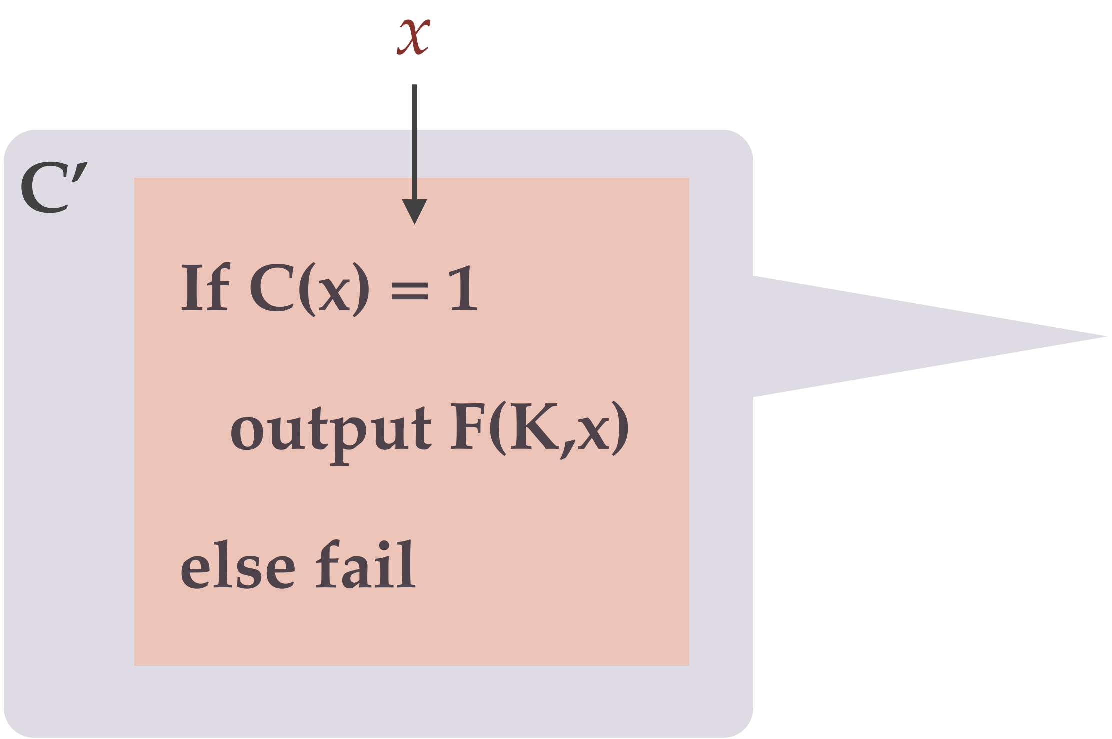
[GGH+'13] [Zimmerman'14] [BGKPS'14]



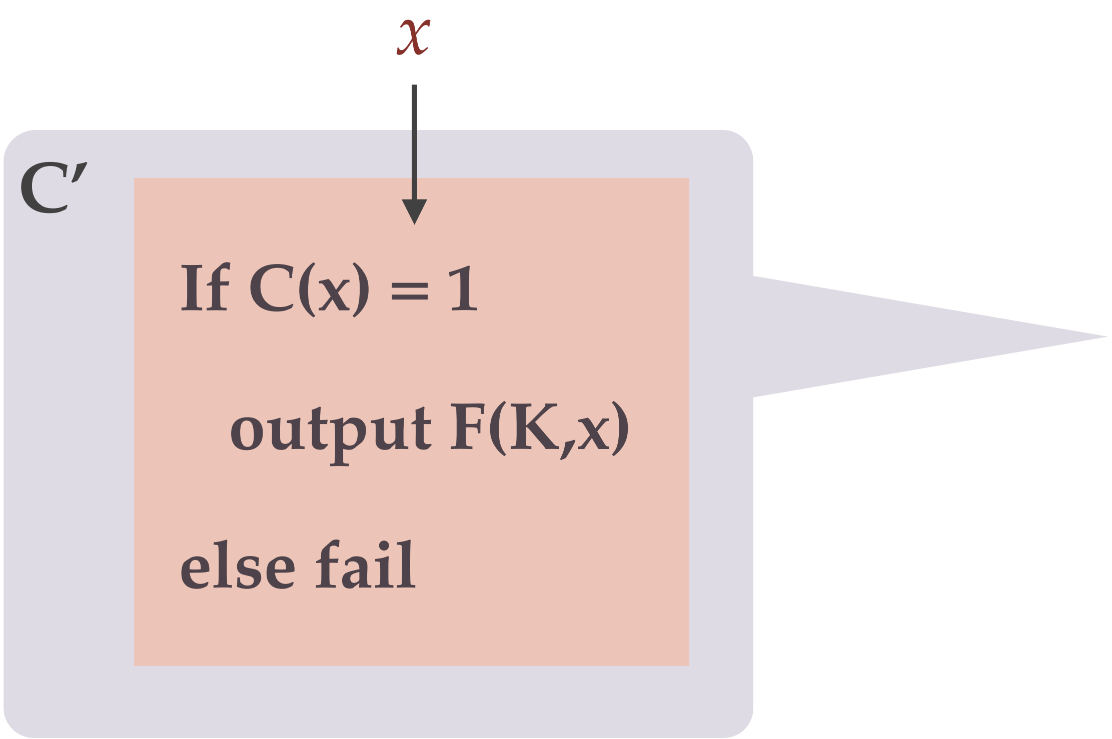
$C_0(x) = C_1(x)$ for all x

CPRF Construction from iO

CPRF Construction from iO



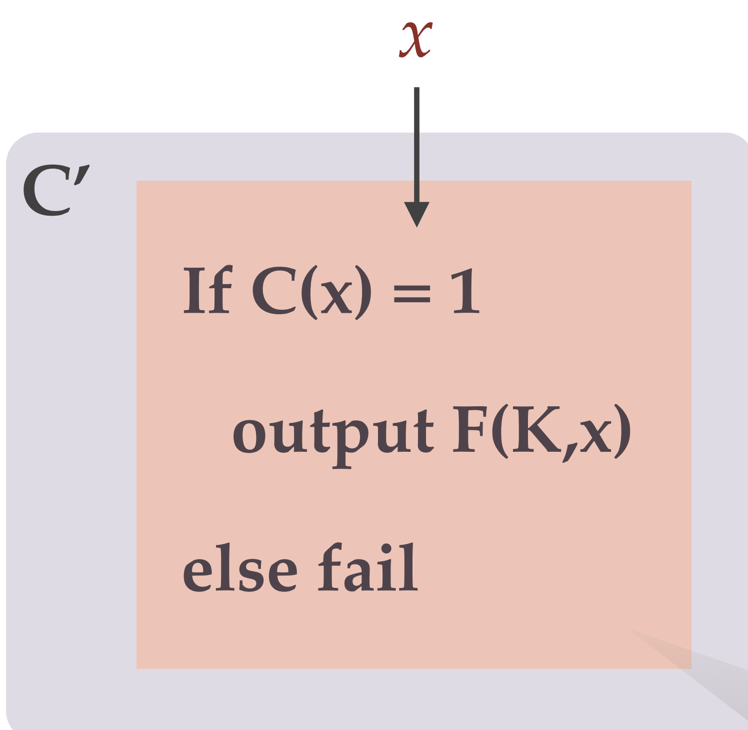
CPRF Construction from iO



Obfuscate the circuit C′

Constrained key = O(C′)

CPRF Construction from iO



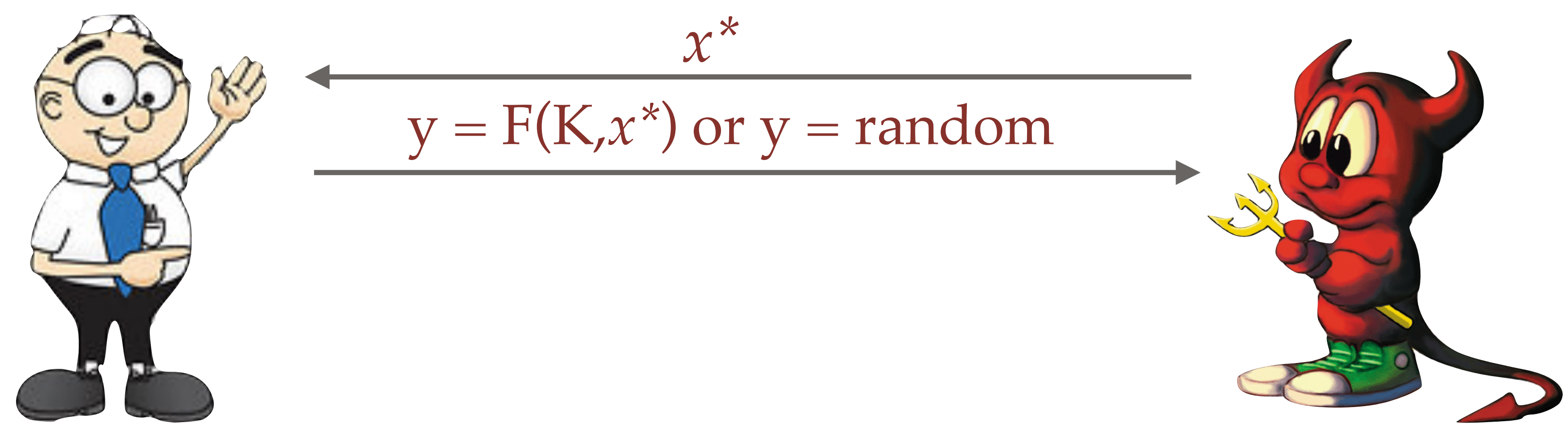
Obfuscate the circuit C′

<u>Constrained key</u> = O(C′)

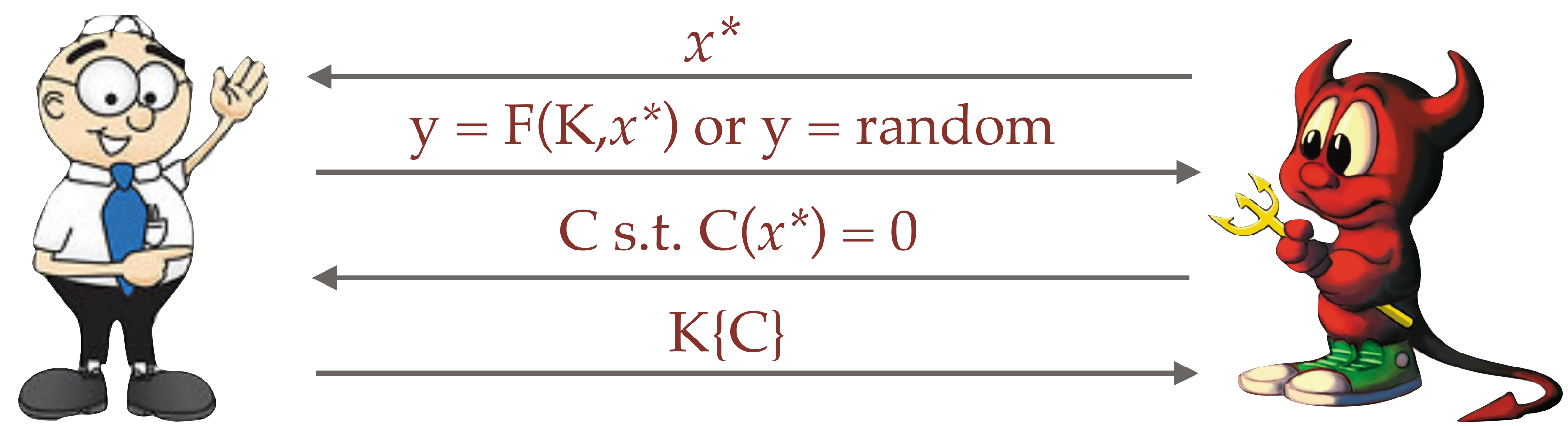
Can prove this construction secure under iO

CPRF Construction: Proof Overview

CPRF Construction: Proof Overview



CPRF Construction: Proof Overview



CPRF Construction: Proof Overview

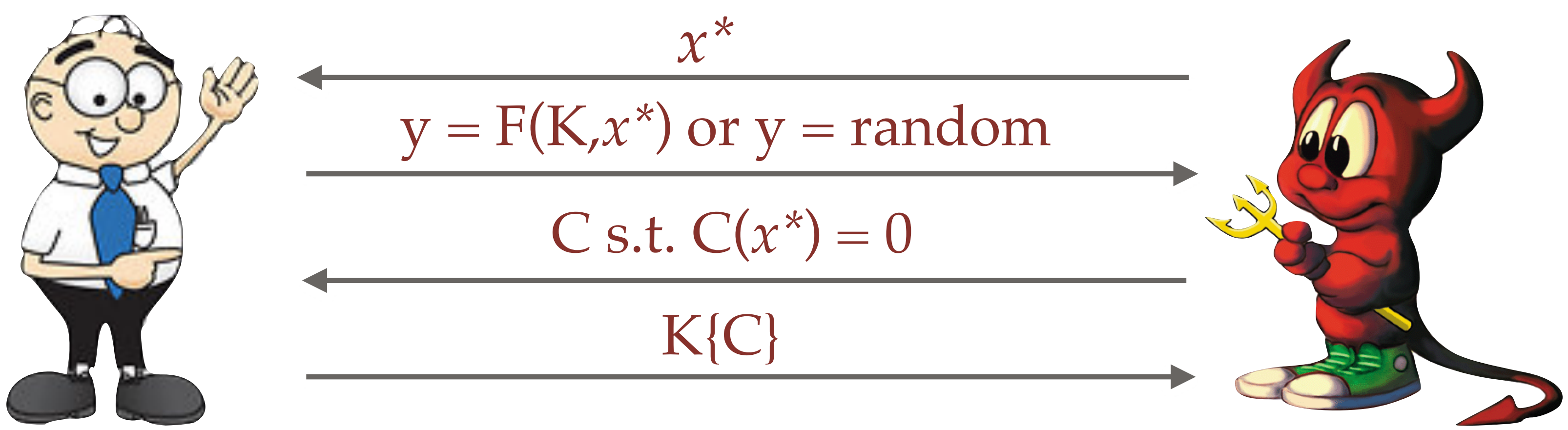
x

K

If C(x) = 1

output F(K,x)

else fail



CPRF Construction: Proof Overview

x

K

If C(x) = 1

output F(K,x)

else fail

x

$K\{x^*\}$

$y^* = F(K,x^*)$

If C(x) = 1

if x = x*

output y*

else output F(K,x)

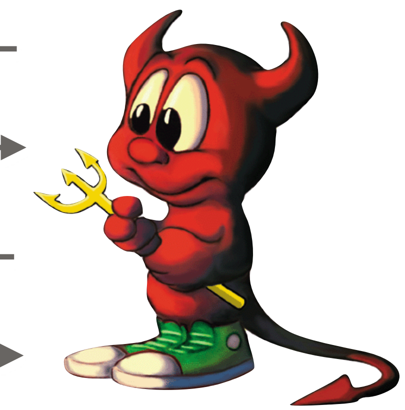
else fail

x^*

$y = F(K,x^*)$ or y = random

C s.t. $C(x^*) = 0$

K{C}



CPRF Construction: Proof Overview

x ↓

K

If C(x) = 1

  output F(K,x)

else fail

⇓ iO

x ↓

$K\{x^*\}$

$y^* = F(K,x^*)$

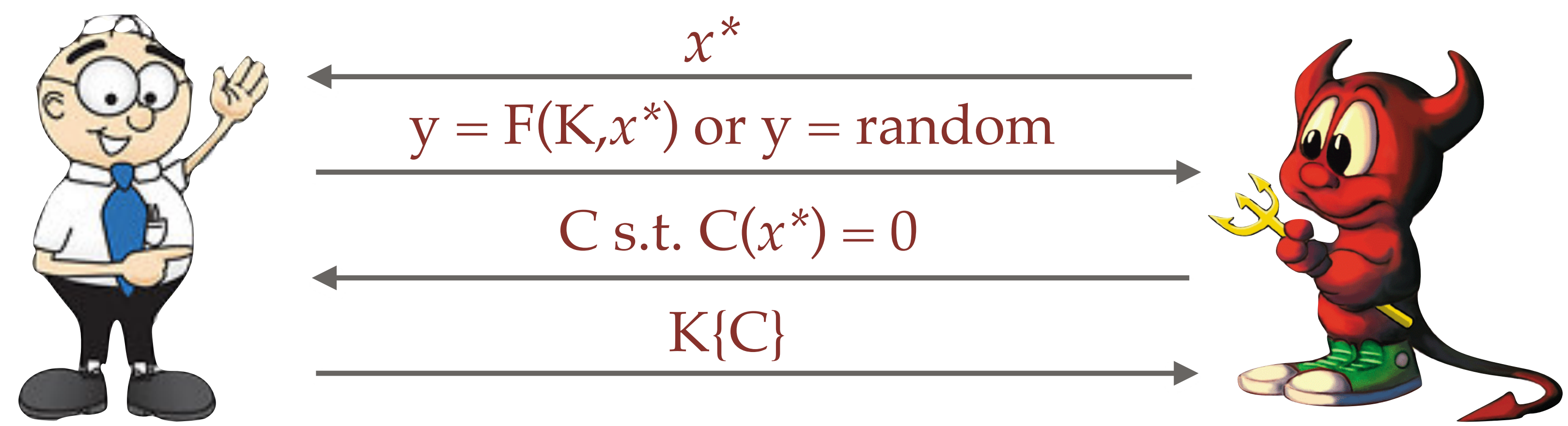
If C(x) = 1

  if x = x*

    output y*

  else output F(K,x)

else fail



CPRF Construction: Proof Overview

x ↓

K

If C(x) = 1

output F(K,x)

else fail

iO ⇓

x ↓

$K\{x^*\}$

$y^* = F(K,x^*)$

If C(x) = 1

if x = x*

output y*

else output F(K,x)

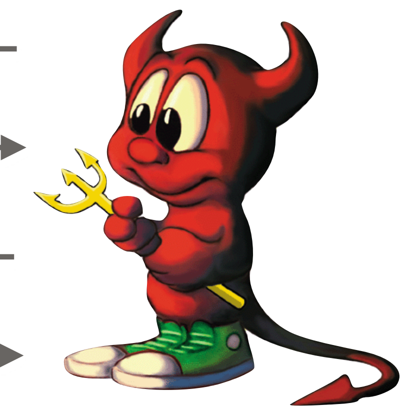
else fail

x^* ←

y = F(K,x^*) or y = random →

C s.t. C(x^*) = 0 ←

K{C} →



⇒

x ↓

$K\{x^*\}$

y^*= *random*

If C(x) = 1

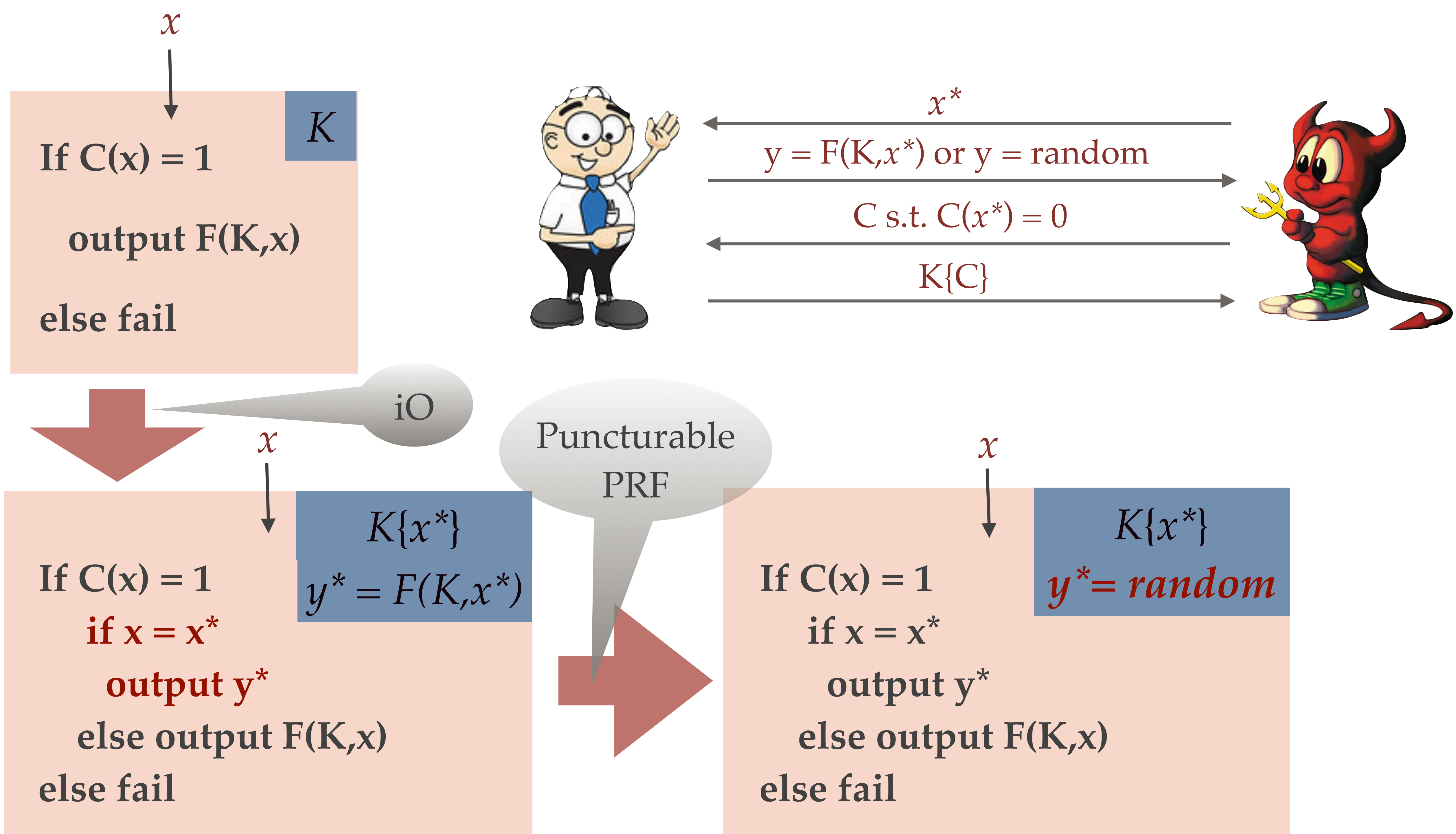
if x = x*

output y*

else output F(K,x)

else fail

CPRF Construction: Proof Overview

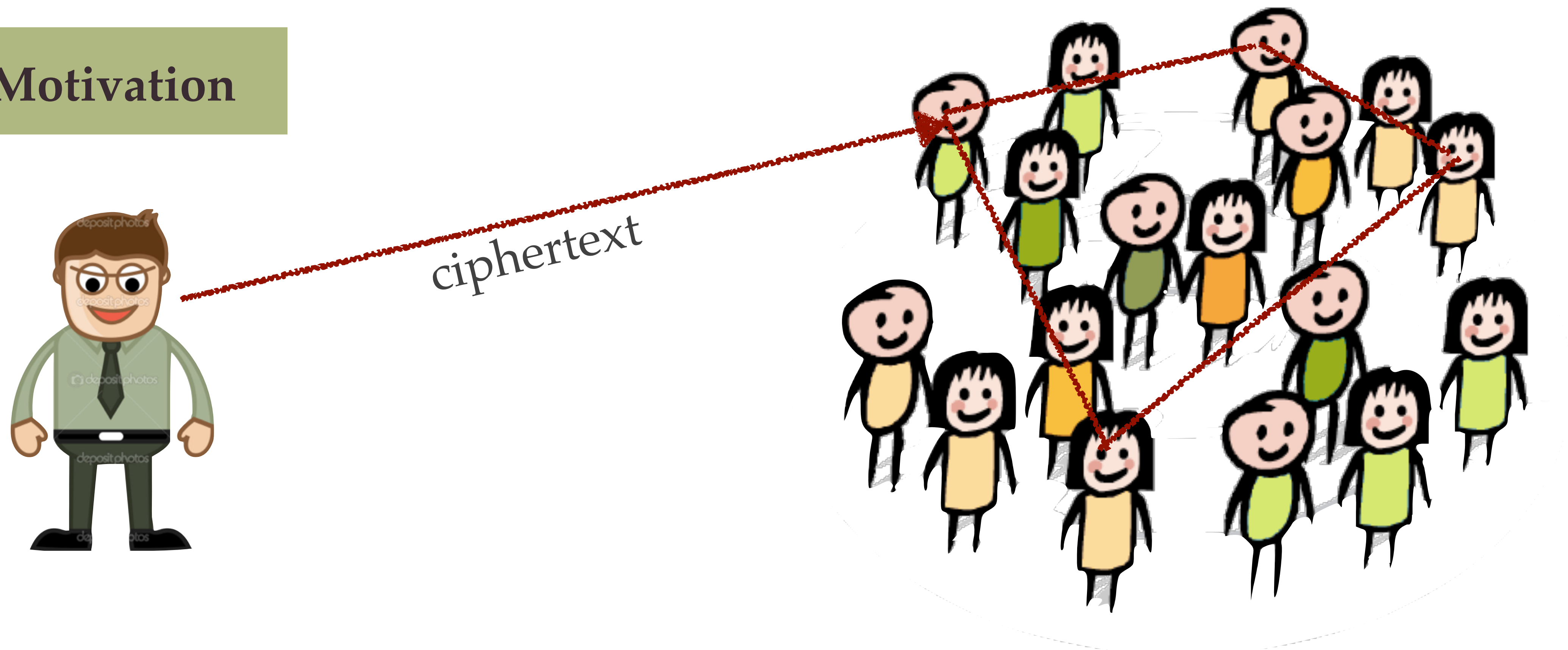


CPRFs for Unbounded Inputs

Motivation

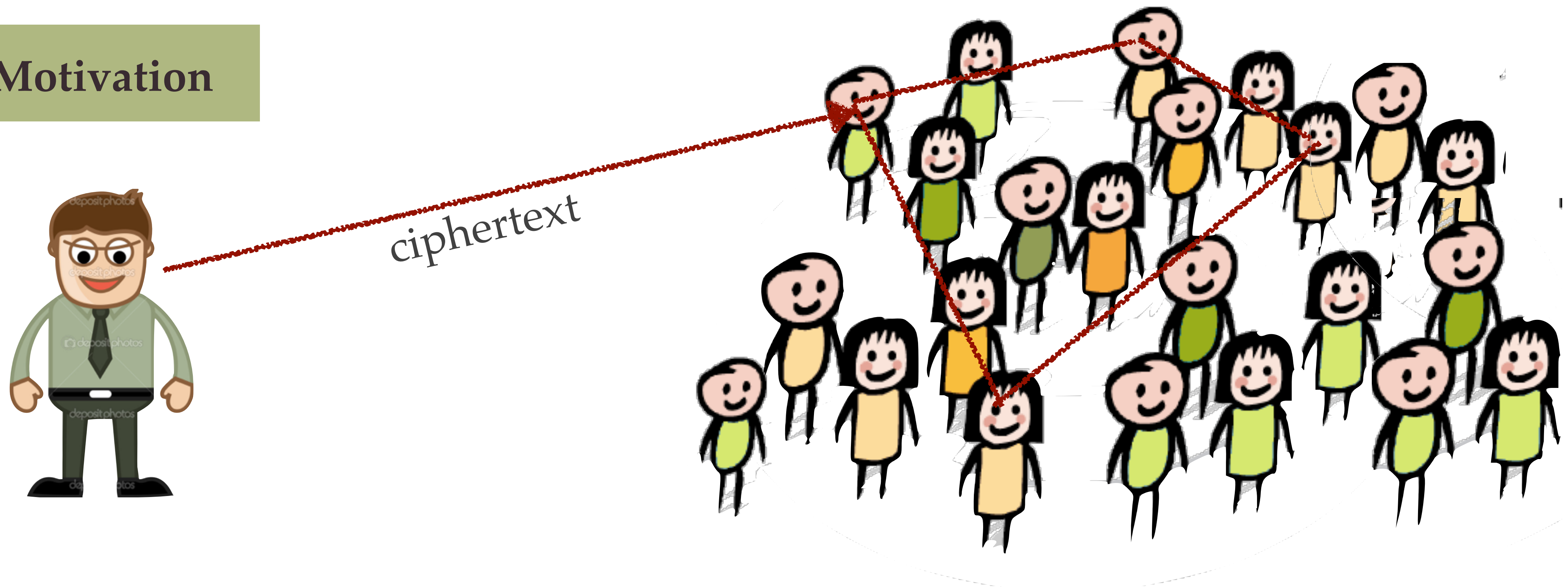
CPRFs for Unbounded Inputs

Motivation



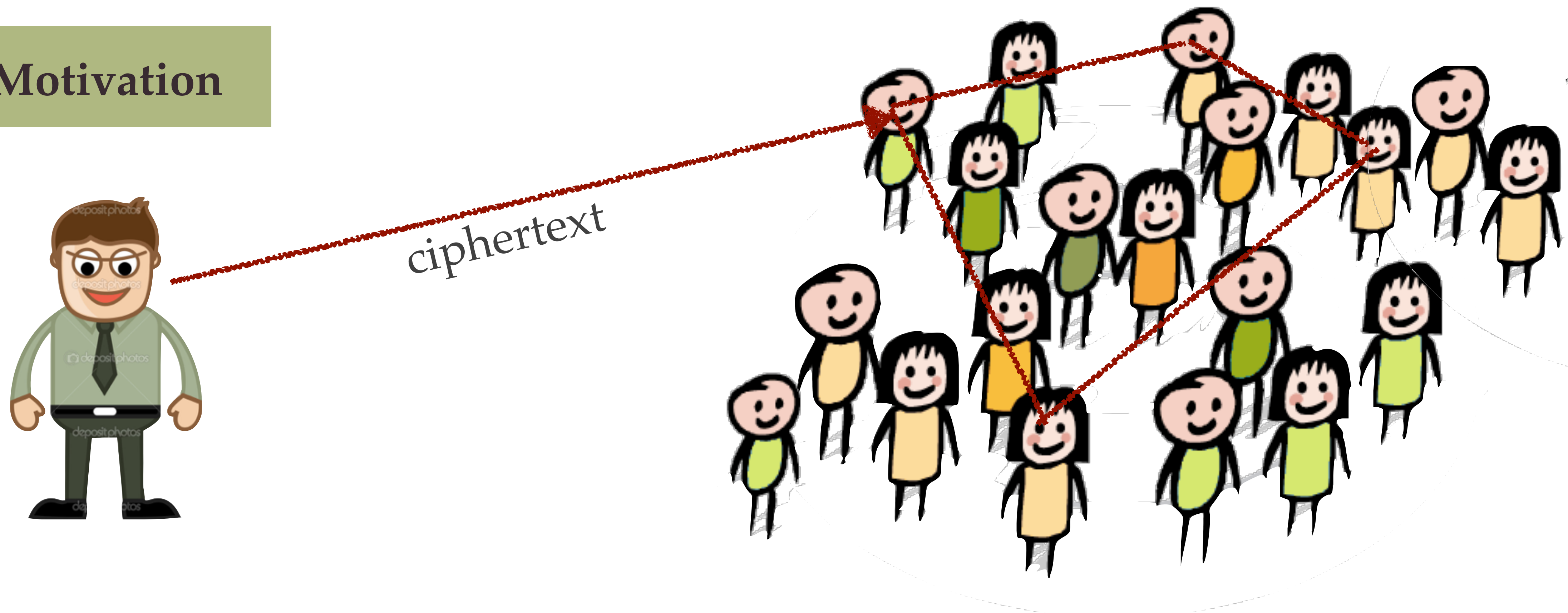
CPRFs for Unbounded Inputs

Motivation



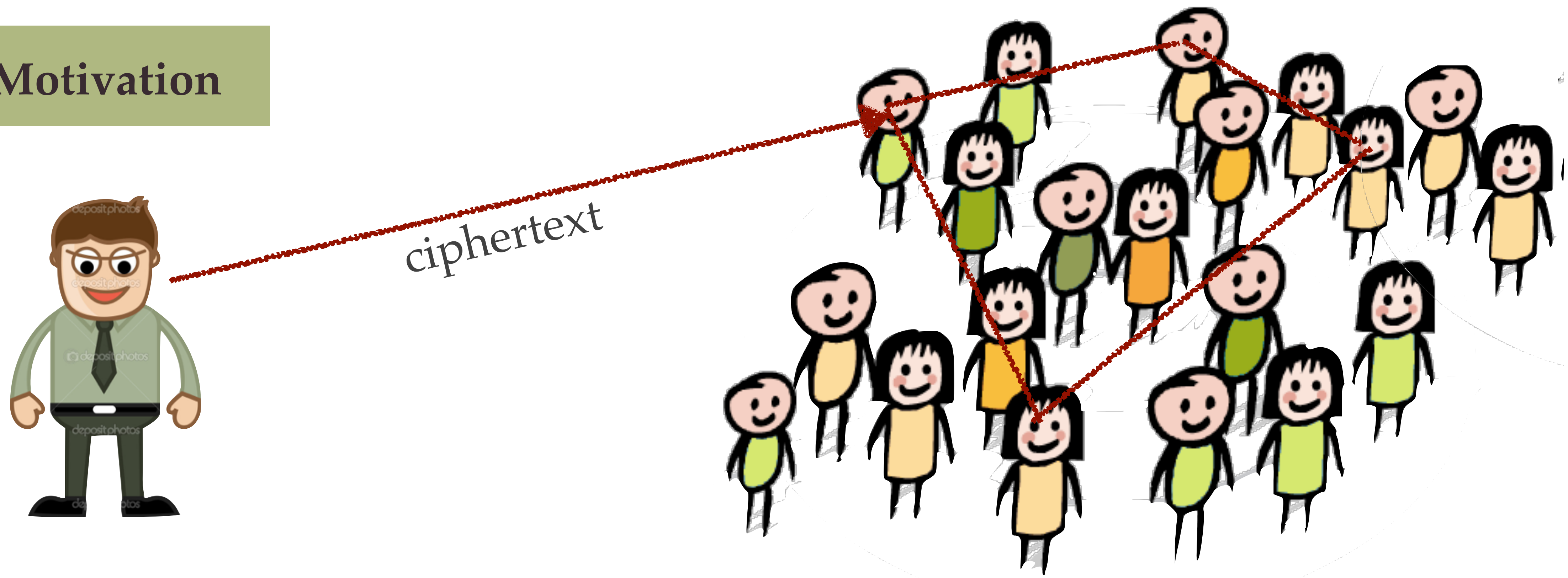
CPRFs for Unbounded Inputs

Motivation



CPRFs for Unbounded Inputs

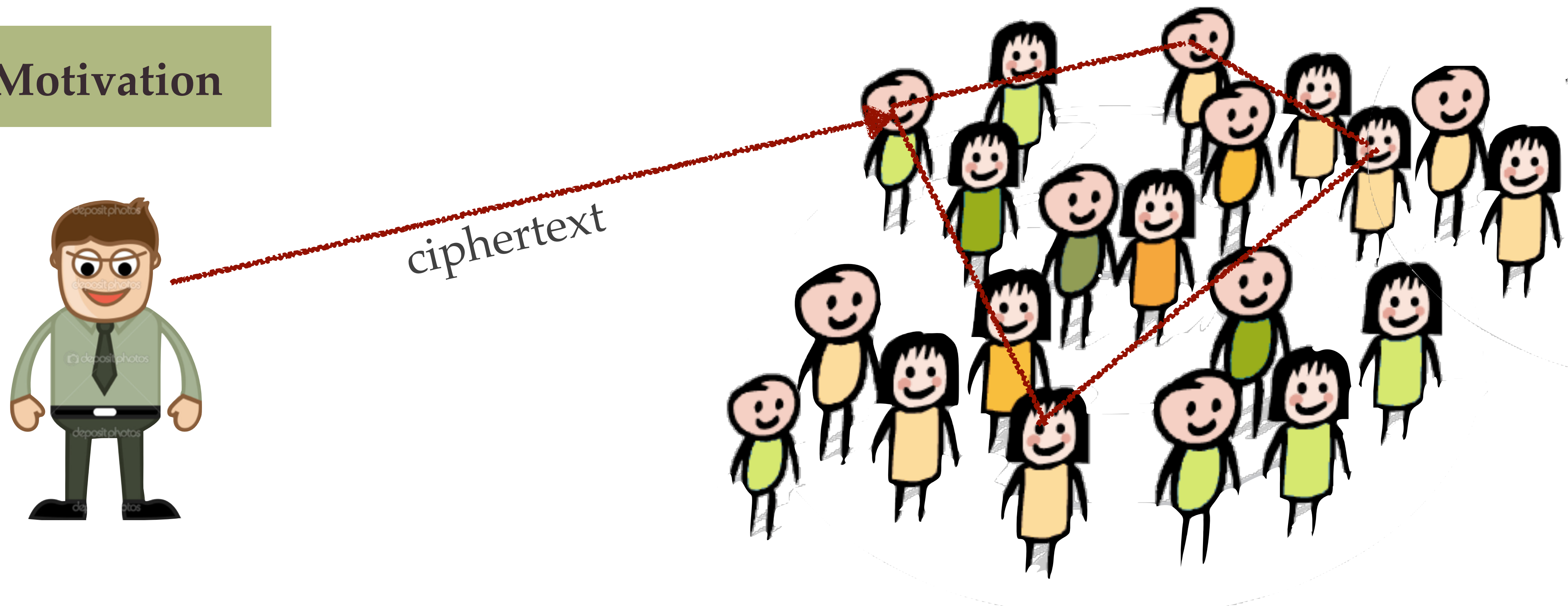
Motivation



Broadcast encryption for unbounded set of users

CPRFs for Unbounded Inputs

Motivation



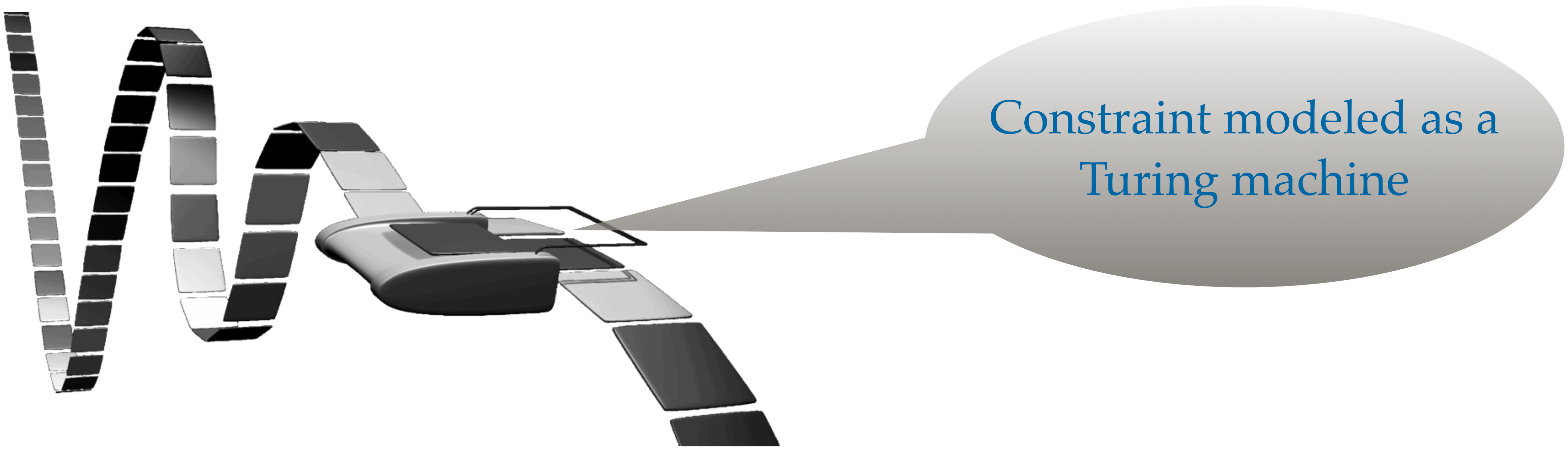
Broadcast encryption for unbounded set of users

- Inputs to the PRF and for the constraint can be of any size, not fixed a priori
- Cannot have constraint as a circuit: Model the constraint as a Turing machine

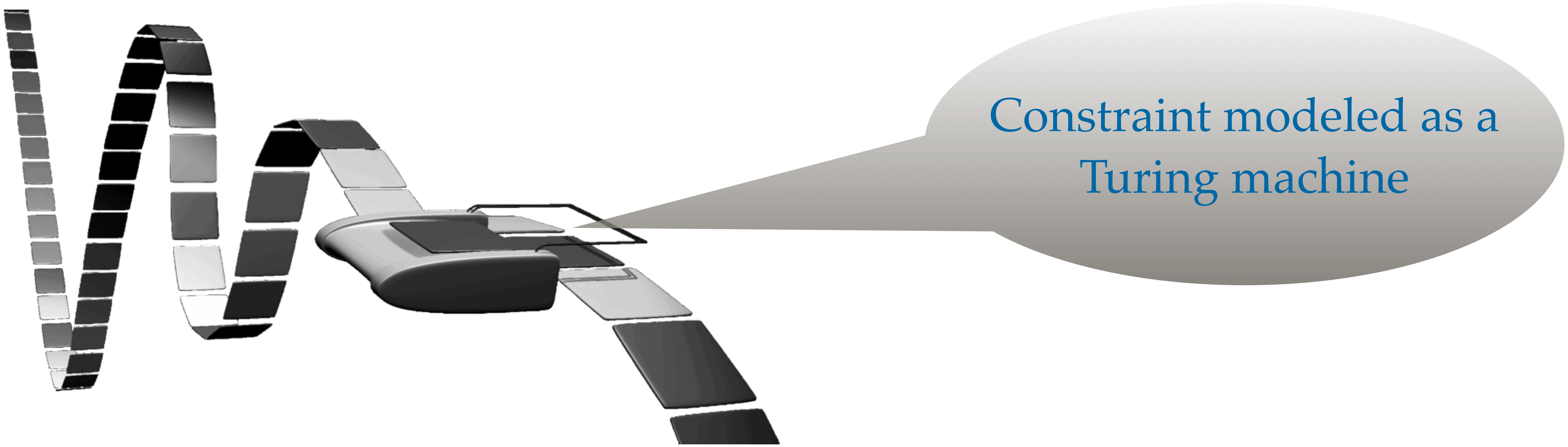
[Zhandry'14]

CPRFs for Unbounded Inputs

CPRFs for Unbounded Inputs



CPRFs for Unbounded Inputs



Results Prior to our Work

- CPRFs for unbounded inputs under the assumption that public-coin differing-inputs obfuscation exists [AFP'14]
- Evidence that differing-inputs obfuscation may not exist [GGHW'14] [BCP'14] [BSW'16]

Our Results

Our Results

Indistinguishability obfuscation for circuits exist

+

Injective Pseudorandom generators exist

Our Results

Indistinguishability obfuscation for circuits exist

+

Injective Pseudorandom generators exist

⬇

Selectively secure constrained PRFs for unbounded inputs exist

Our Results

Indistinguishability obfuscation for circuits exist

+

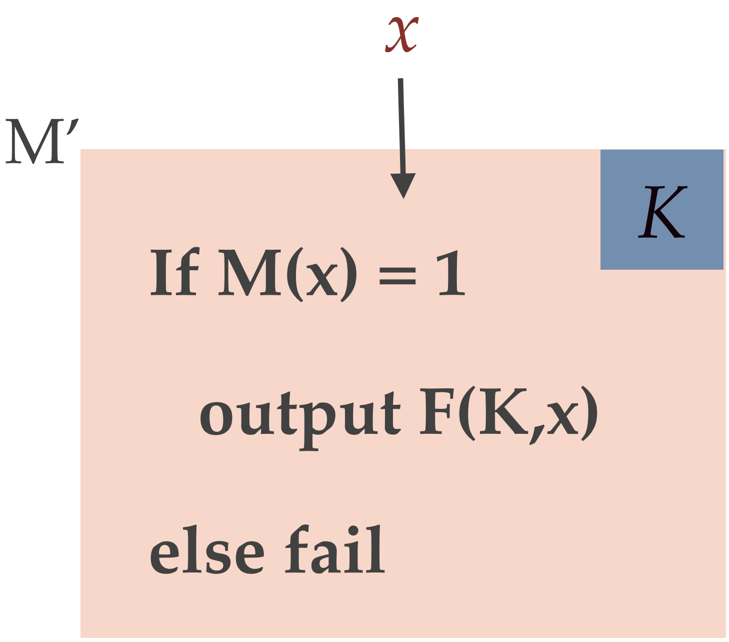
Injective Pseudorandom generators exist

Selectively secure constrained PRFs for unbounded inputs exist

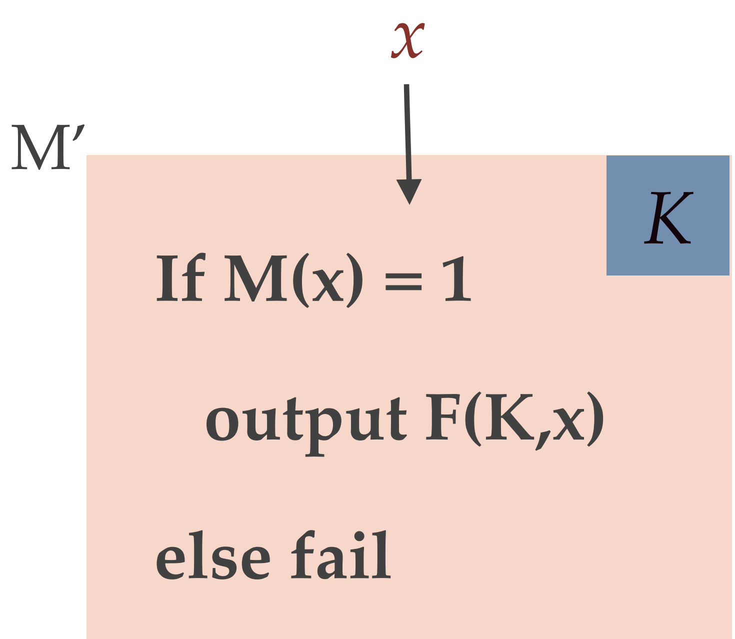
Attribute-based Encryption for Turing Machines

CPRF Construction: Intuition

CPRF Construction: Intuition

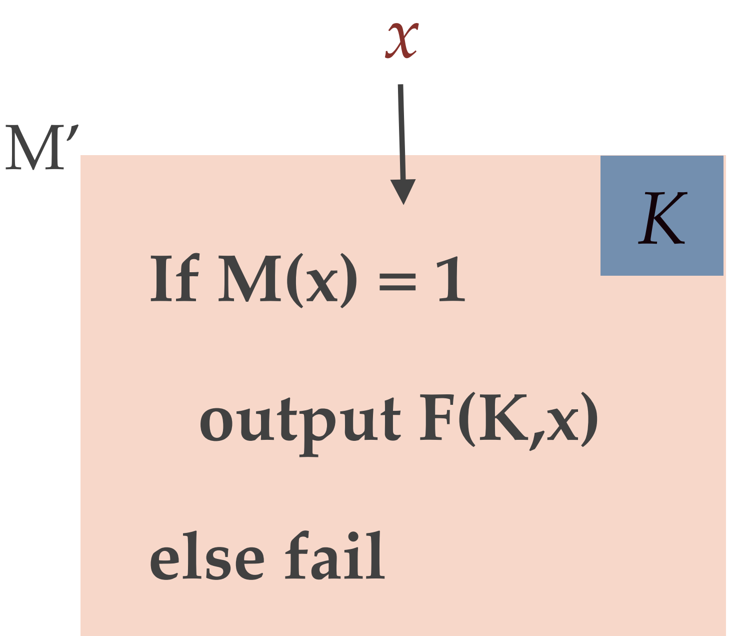


CPRF Construction: Intuition



Constraint expressed as Turing machine:
Why not obfuscate this Turing machine ?

CPRF Construction: Intuition

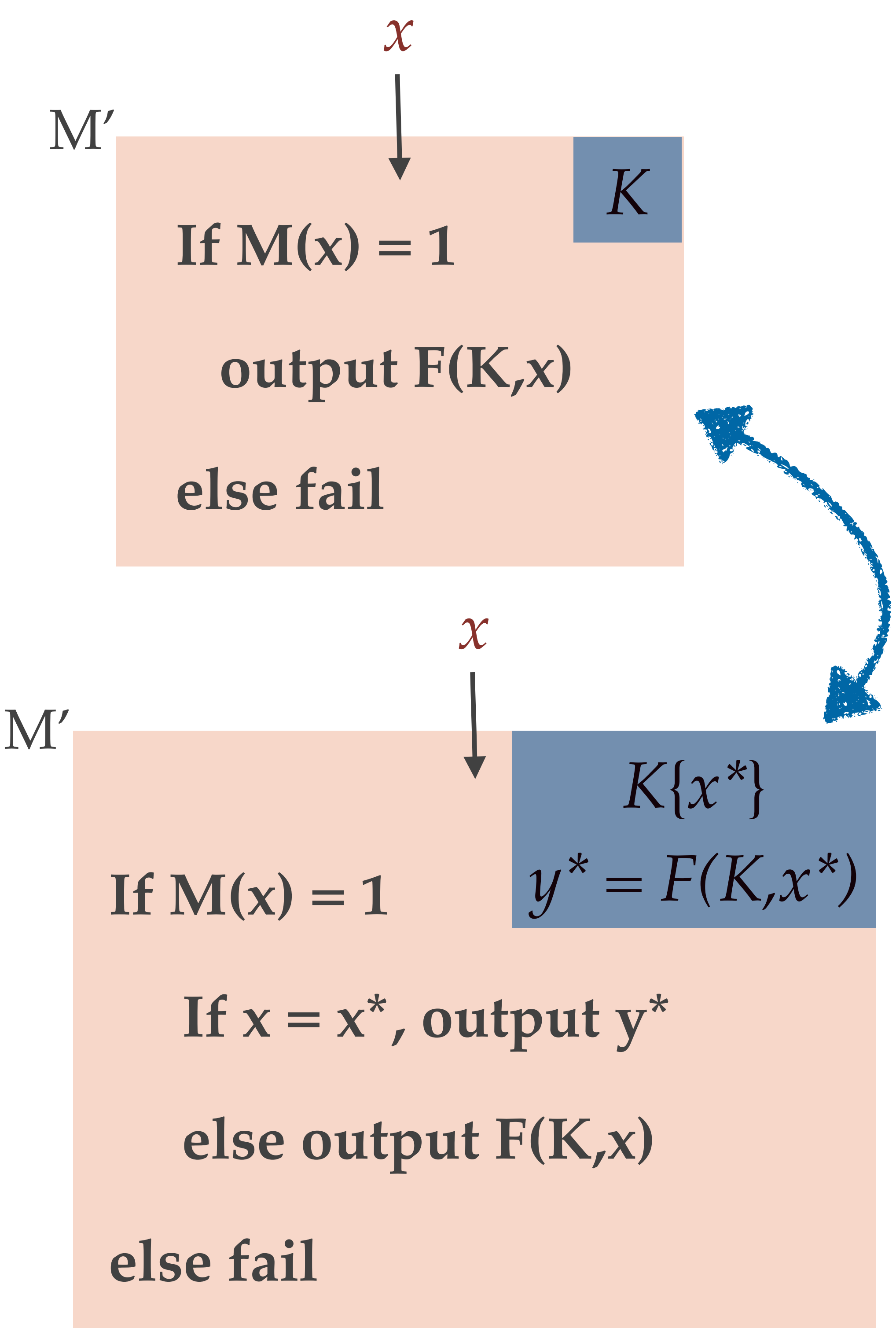


Constraint expressed as Turing machine:
Why not obfuscate this Turing machine ?

<u>iO for Turing machines:</u> Size of obfuscated TM depends on |M'|, requires a priori bound on |M'| and on input sizes

[KLW'14]

CPRF Construction: Intuition

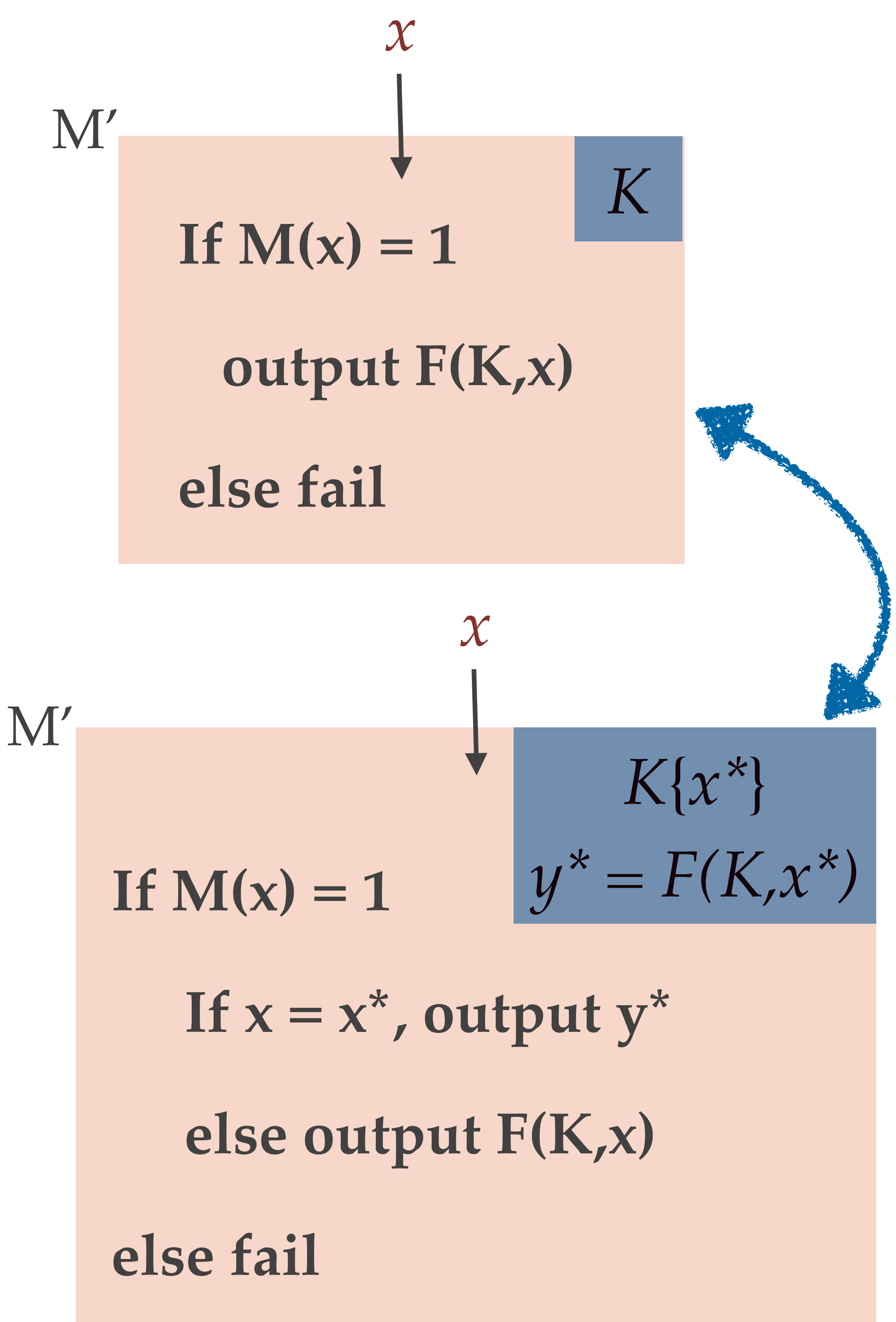


Constraint expressed as Turing machine:
Why not obfuscate this Turing machine ?

<u>iO for Turing machines:</u> Size of obfuscated TM depends on |M′|, requires a priori bound on |M′| and on input sizes

[KLW’14]

CPRF Construction: Intuition



Constraint expressed as Turing machine:
Why not obfuscate this Turing machine ?

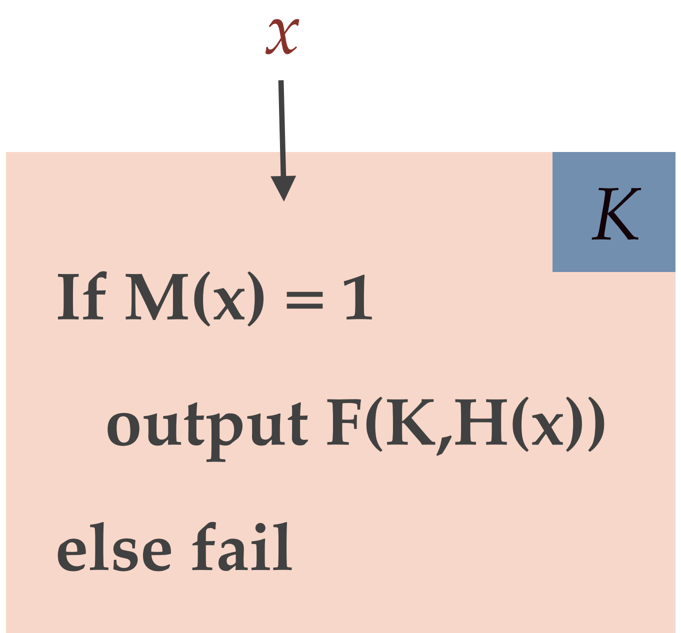
iO for Turing machines: Size of obfuscated TM depends on | M′ |, requires a priori bound on | M′ | and on input sizes

[KLW'14]

Difficult to make this switch without knowing | x* |

CPRF Construction: Intuition

CPRF Construction: Intuition



What if we map x to fixed sized strings?

Use F(K, H(x)) instead of F(K,x)?

CPRF Construction: Intuition

x

K

If M(x) = 1

output F(K,H(x))

else fail

What if we map x to fixed sized strings?

Use F(K, H(x)) instead of F(K,x)?

x

$K\{H(x^*) = h^*\}$

$y^* = F(K,h^*)$

If M(x) = 1
if H(x) = h*
output y*
else output F(K,H(x))
else fail

x

$K\{H(x^*) = h^*\}$

$y^* = random$

If M(x) = 1
if H(x) = h*
output y*
else output F(K,H(x))
else fail

CPRF Construction: Intuition

What if we map x to fixed sized strings?

Use F(K, H(x)) instead of F(K,x)?

x

K

If M(x) = 1

output F(K,H(x))

else fail

<u>iO fails:</u> u,v such that H(u) = H(v)
but M(u) = 1 and M(v) = 0

x

$K\{H(x^*)= h^*\}$

$y^* = F(K,h^*)$

If M(x) = 1
if H(x) = h*
output y*
else output F(K,H(x))
else fail

x

$K\{H(x^*)= h^*\}$

$y^* = random$

If M(x) = 1
if H(x) = h*
output y*
else output F(K,H(x))
else fail

CPRF Construction: Intuition

What if we map x to fixed sized strings?

Use F(K, H(x)) instead of F(K,x)?

x

K

If M(x) = 1

output F(K,H(x))

else fail

iO fails: u,v such that H(u) = H(v)
but M(u) = 1 and M(v) = 0

x

$K\{H(x^*)= h^*\}$

$y^* = F(K,h^*)$

If M(x) = 1
if H(x) = h*
output y*
else output F(K,H(x))
else fail

output for x = u: F(K,h*)

x

$K\{H(x^*)= h^*\}$

$y^* = random$

If M(x) = 1
if H(x) = h*
output y*
else output F(K,H(x))
else fail

output for x = u: random

CPRF Construction: Intuition

<u>If we use some hash function:</u>

Crucial that no with input with $H(x) = h^*$ enters this region

$K\{H(x^*) = h^*\}$
$y^* = F(K, h^*)$

If M(x) = 1

if H(x) = h*

output y*

else output F(K,H(x))

else fail

CPRF Construction: Intuition

<u>If we use some hash function:</u>

Crucial that no with input with $H(x) = h^*$ enters this region

Need "iO-friendly" hash-function

If M(x) = 1

if H(x) = h*

output y*

else output F(K,H(x))

else fail

$K\{H(x^*) = h^*\}$

$y^* = F(K, h^*)$

CPRF Construction: Intuition

<u>If we use some hash function:</u>

Crucial that no with input with $H(x) = h^*$ enters this region

Need "iO-friendly" hash-function

Positional Accumulator

[KLW'14]

$K\{H(x^*) = h^*\}$

$y^* = F(K, h^*)$

If M(x) = 1

if H(x) = h*

output y*

else output F(K,H(x))

else fail

CPRF Construction: Intuition

<u>If we use some hash function:</u>

Crucial that no with input with $H(x) = h^*$ enters this region

Need "iO-friendly" hash-function

Positional Accumulator

[KLW'14]

If M(x) = 1

if $H(x) = h^*$

output y^*

else output F(K,H(x))

else fail

$K\{H(x^*)= h^*\}$

$y^* = F(K,h^*)$

Hash the value on tape

PA(x) = acc

<u>PRF Eval:</u> F(k,acc)

CPRF Construction: Intuition

<u>If we use some hash function:</u>

Crucial that no with input with
$H(x) = h^*$ enters this region

Need "iO-friendly" hash-function

Positional Accumulator

[KLW'14]

- Succinct Verifiability
- Succinct Updatability
- Selective Enforcement

$K\{H(x^*)= h^*\}$
$y^* = F(K,h^*)$

If M(x) = 1
  if H(x) = h*
    output y*
  else output F(K,H(x))
else fail

Hash the value on tape
PA(x) = acc
<u>PRF Eval:</u> F(k,acc)

CPRF Construction: Intuition

<u>If we use some hash function:</u>

Crucial that no with input with
$H(x) = h^*$ enters this region

Need "iO-friendly" hash-function

Positional Accumulator

[KLW'14]

- Succinct Verifiability
- Succinct Updatability
- <u>Selective Enforcement</u>

$K\{H(x^*) = h^*\}$
$y^* = F(K, h^*)$

If M(x) = 1

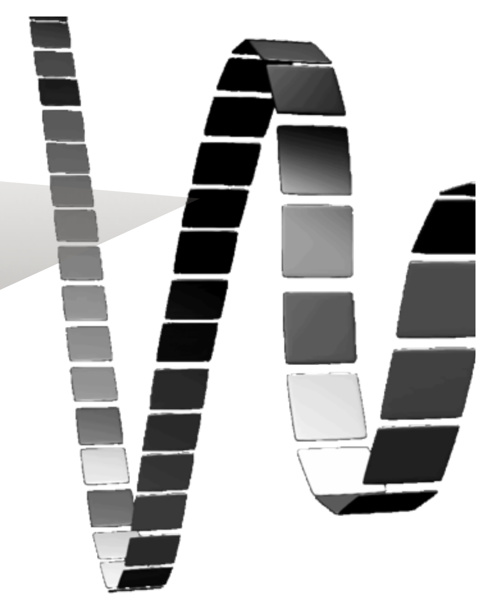
if H(x) = h*

output y*

else output F(K,H(x))

else fail

Hash the value on tape
PA(x) = acc
<u>PRF Eval:</u> F(k,acc)



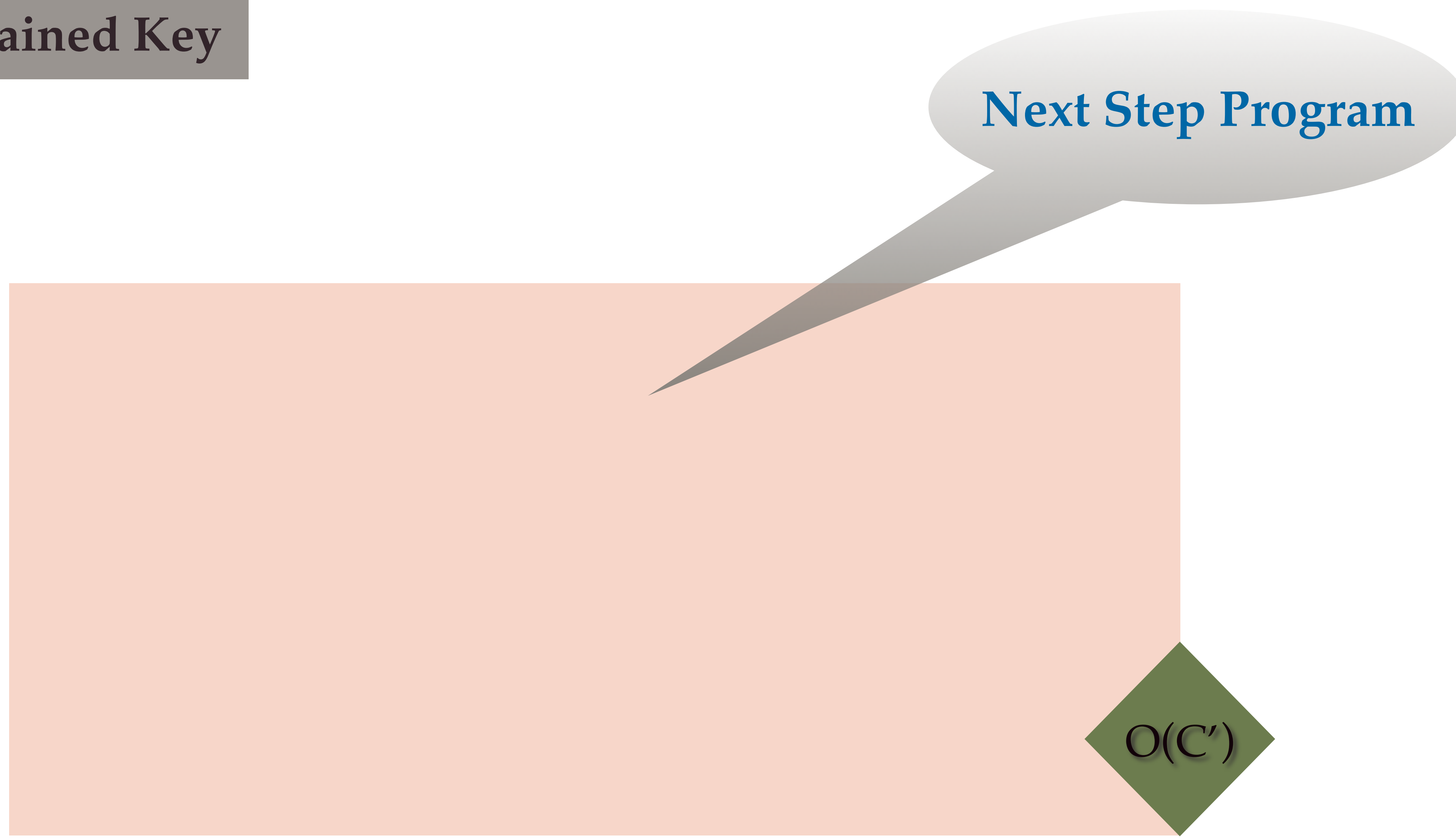
Construction Overview

Construction Overview

PRF Constrained Key

Construction Overview

PRF Constrained Key



Construction Overview

PRF Constrained Key

Next Step Program

$(i, st_i, sym_i, pos_i, acc)$

k,*M*

- Compute $t(st_i, sym_i) = (st_{i+1}, sym_{i+1}, pos_{i+1})$
- If st_{i+1} is a reject state, output **fail**

  else if st_{i+1} is accept, output **F(*k*,acc)**

  else output **(st_{i+1}, sym_{i+1}, pos_{i+1})**

O(C')

Construction Overview

PRF Constrained Key

Next Step Program

$(i, st_i, sym_i, pos_i, acc)$

k,*M*

- Compute $t(st_i, sym_i) = (st_{i+1}, sym_{i+1}, pos_{i+1})$
- If st_{i+1} is a reject state, output **fail**

  else if st_{i+1} is accept, output **F(*k*,acc)**

  else output $\mathbf{(st_{i+1}, sym_{i+1}, pos_{i+1})}$

O(C')

Construction Overview

PRF Constrained Key

Next Step Program

$(i, st_i, sym_i, pos_i, acc)$

k,*M*

- Compute $t(st_i, sym_i) = (st_{i+1}, sym_{i+1}, pos_{i+1})$
- If st_{i+1} is a reject state, output **fail**

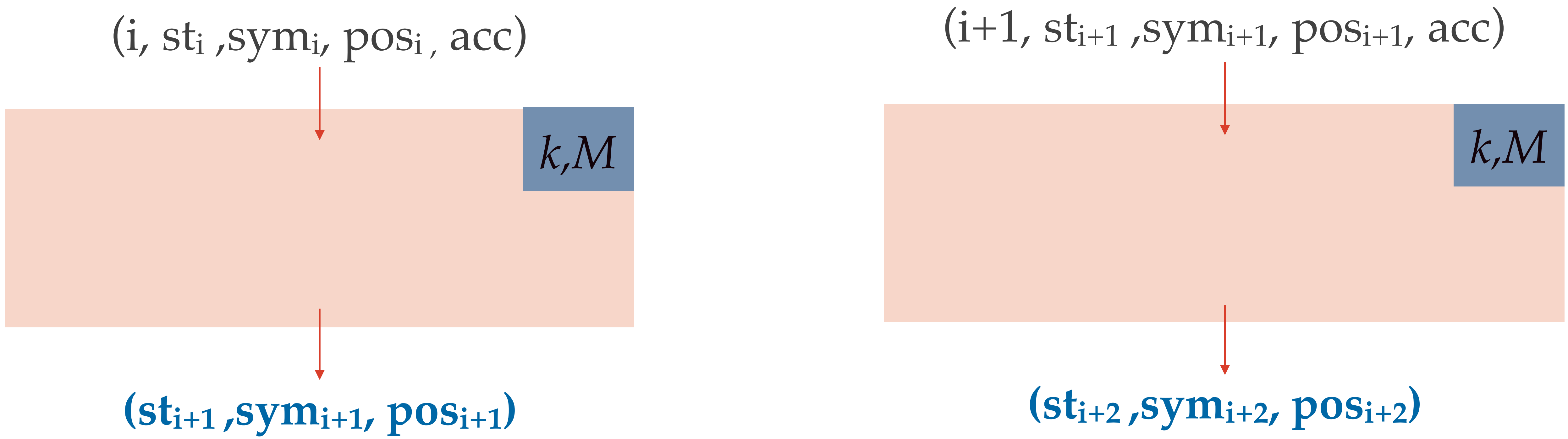
  else if st_{i+1} is accept, output **F(*k*,acc)**

  else output **$(st_{i+1}, sym_{i+1}, pos_{i+1})$**

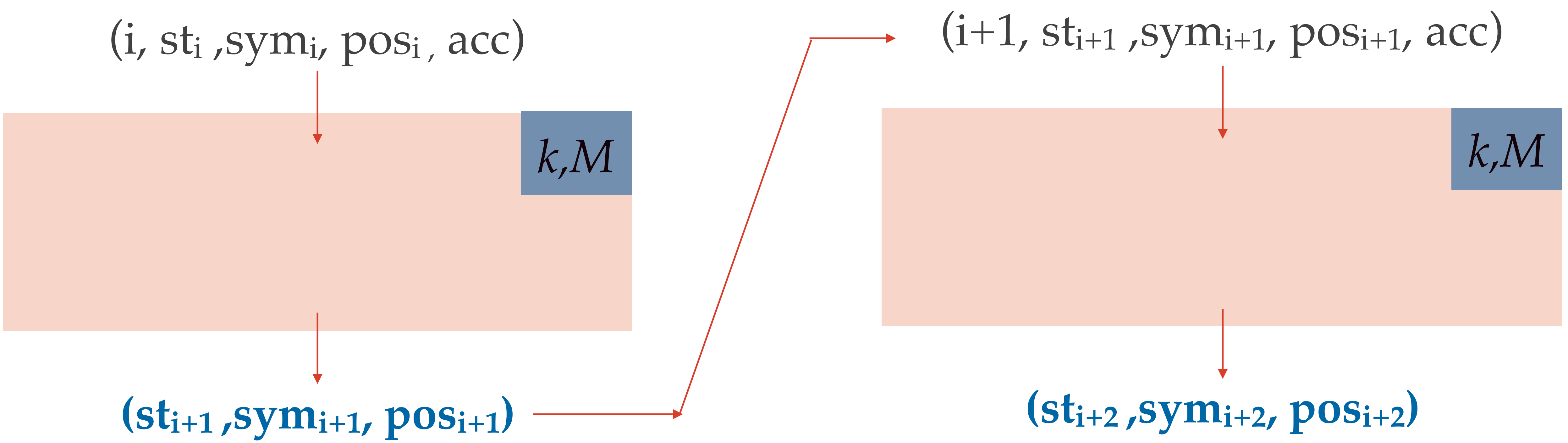
O(C')

Issue: User can input illegal states, symbols to this program

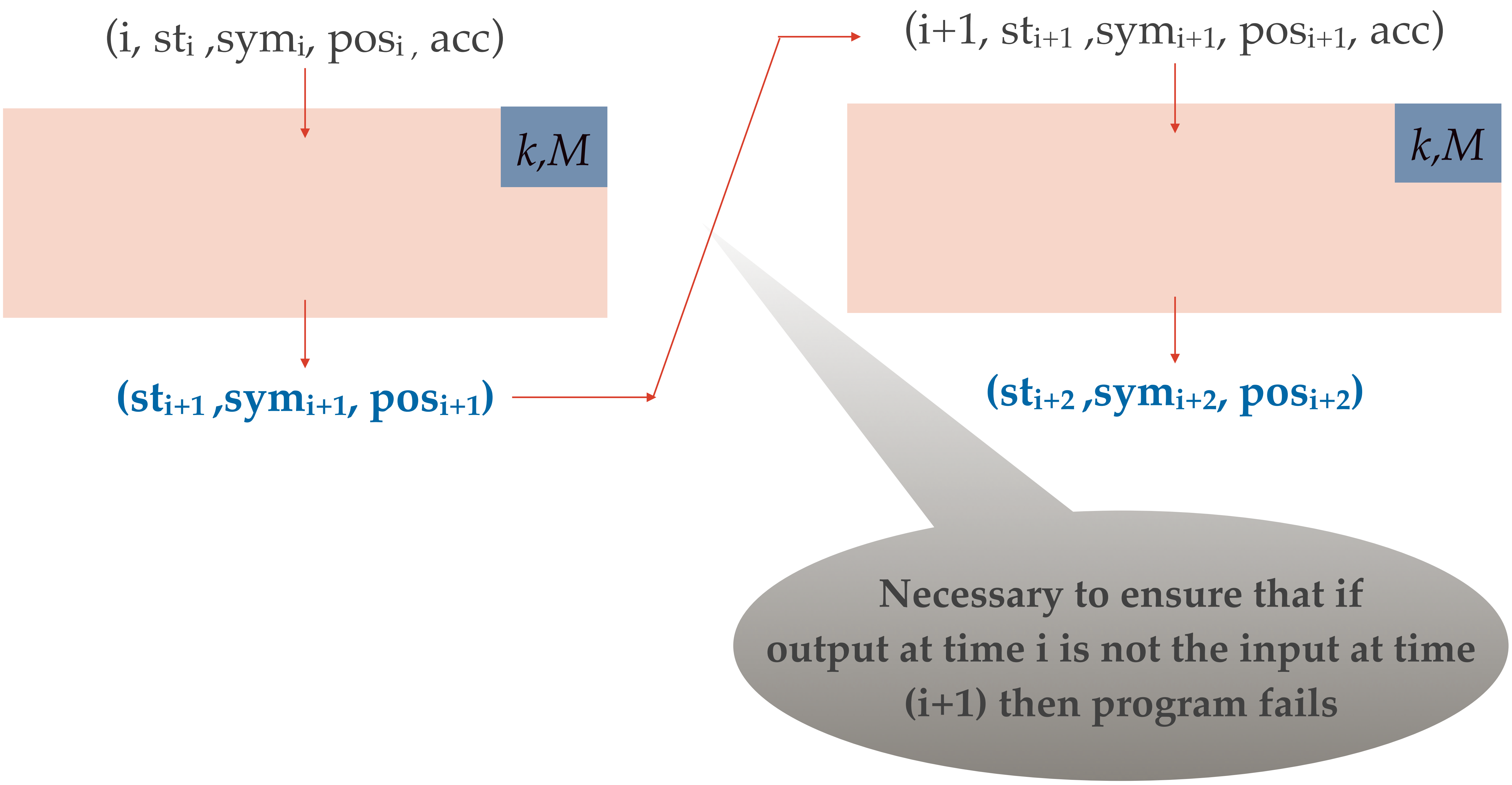
Construction Overview



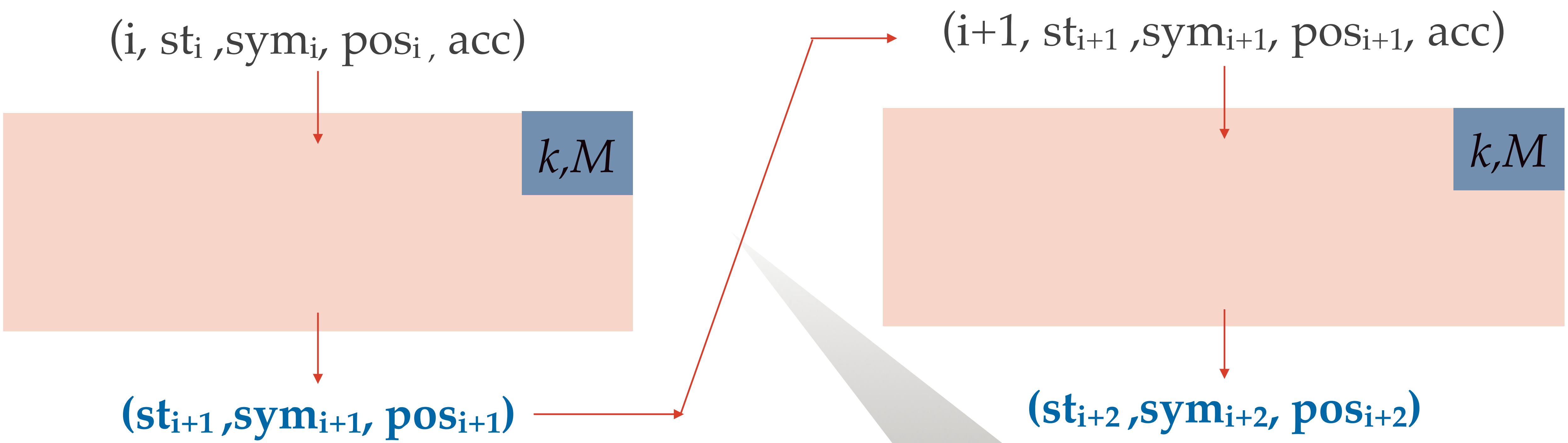
Construction Overview



Construction Overview



Construction Overview



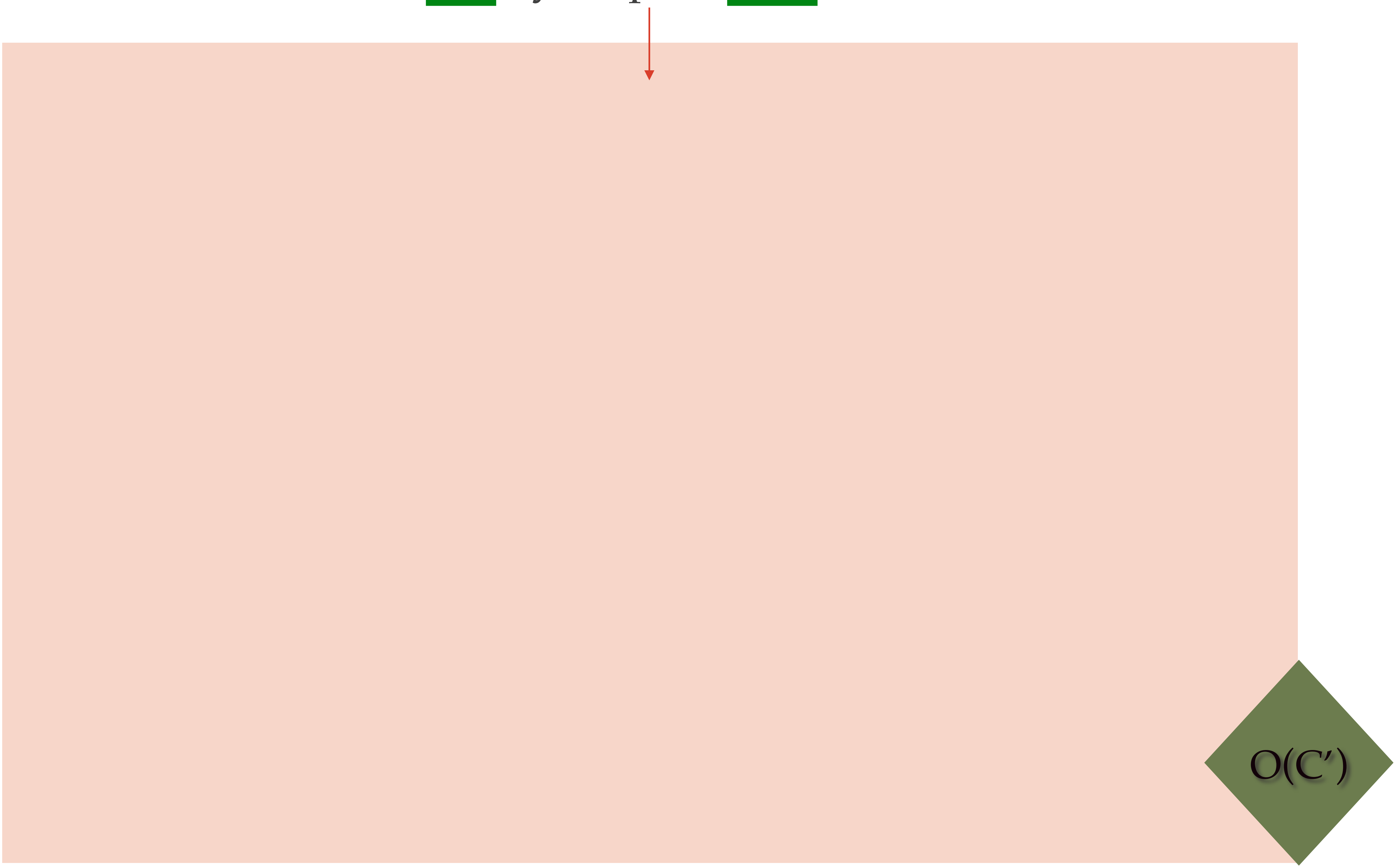
Necessary to ensure that if output at time i is not the input at time (i+1) then program fails

[KLW'14]

- Use ``Splittable'' Signatures !
- Use Accumulator at each step

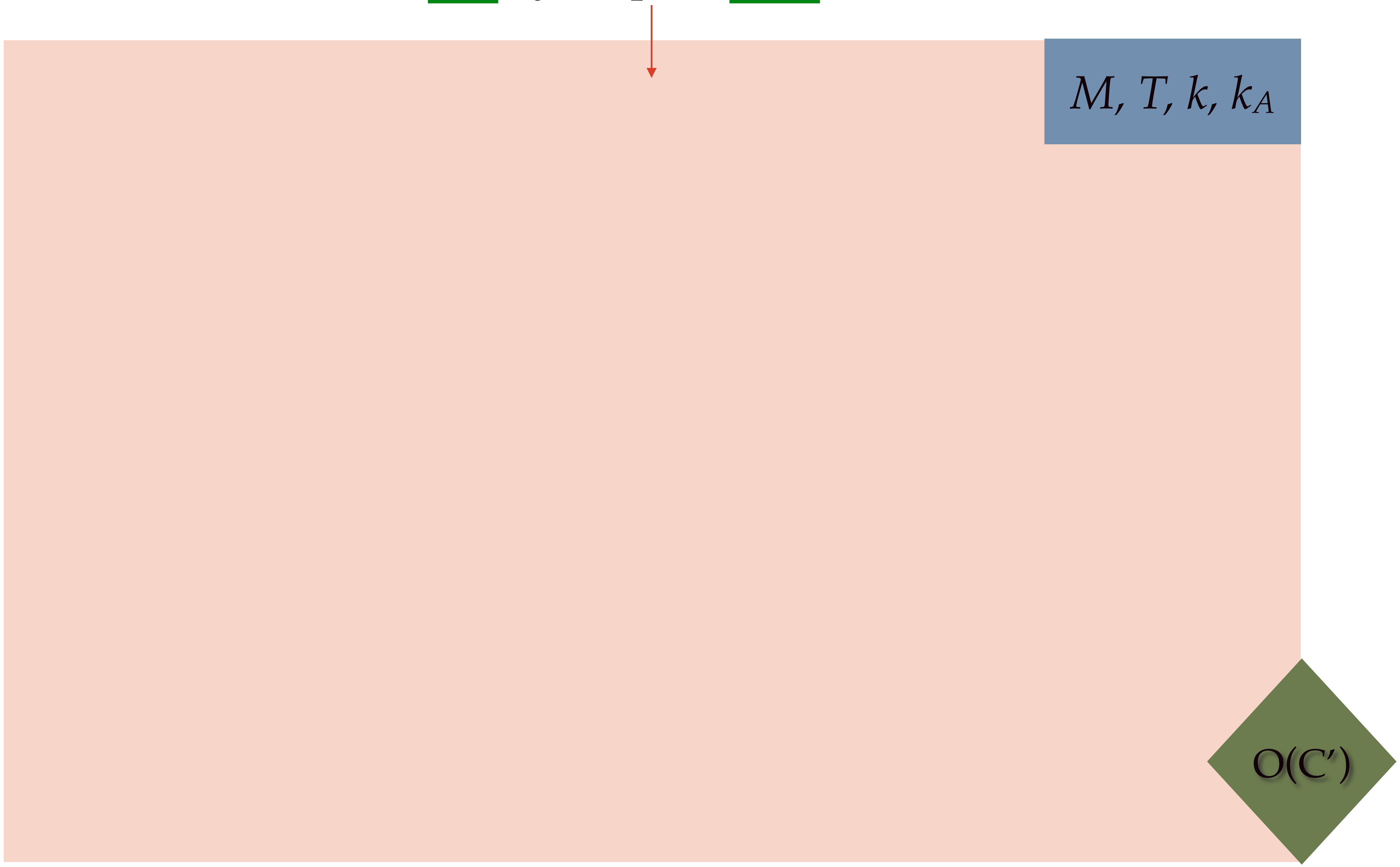
CPRF Construction: Almost Final

$(t, st_i, \underline{sig}, sym_i, pos_i, \underline{acc_i}, acc)$



CPRF Construction: Almost Final

$(t, st_i, \underline{sig}, sym_i, pos_i, \underline{acc_i}, acc)$



CPRF Construction: Almost Final

(t, st_i , sig, sym_i, pos_i , acc_i, acc)

M, T, k, k_A

- **Verify Current Accumulator Value** acc_i, else **fail**

O(C')

CPRF Construction: Almost Final

$(t, st_i, sig, sym_i, pos_i, acc_i, acc)$

M, T, k, k_A

- **Verify Current Accumulator Value** acc_i, else **fail**
- $r = F(k_A, (acc, t - 1))$; $(SK_A, VK_A) = Setup(1^k; r_A)$
  Verify Signature on (st_i, pos_i, acc_i), else **fail**

O(C')

CPRF Construction: Almost Final

(t, st_i, sig, sym_i, pos_i, acc_i, acc)

M, T, k, k_A

- **Verify Current Accumulator Value** acc_i, else **fail**
- $r = F(k_A, (acc, t-1))$; $(SK_A, VK_A) = Setup(1^k; r_A)$
  Verify Signature on (st_i, pos_i, acc_i), else **fail**
- Compute $t(st_i, sym_i) = (st_{i+1}, sym_{i+1}, pos_{i+1})$

  If st_{i+1} is a reject state, output **fail**
  else if st_{i+1} is accept, output **F(*k*,acc)**

Next Step

O(C')

CPRF Construction: Almost Final

(t, st_i ,sig, sym_i, pos_i , acc_i, acc)

M, T, k, k_A

- **Verify Current Accumulator Value** acc_i, else **fail**
- $r = F(k_A, (acc, t-1))$; $(SK_A, VK_A) = Setup(1^k; r_A)$
  Verify Signature on (st_i, pos_i , acc_i), else **fail**
- Compute $t(st_i, sym_i) = (st_{i+1}, sym_{i+1}, pos_{i+1})$

  If st_{i+1} is a reject state, output **fail**
  else if st_{i+1} is accept, output **F(*k*,acc)**

  Next Step
- acc_{i+1} = **Update Accumulator**(pos_i , sym_{i+1}, acc_i)

O(C')

CPRF Construction: Almost Final

$(t, st_i, \underline{sig}, sym_i, pos_i, \underline{acc_i}, acc)$

M, T, k, k_A

- **Verify Current Accumulator Value** acc_i, else **fail**
- $r = F(k_A, (acc, t-1))$; $(SK_A, VK_A) = Setup(1^k; r_A)$
  Verify Signature on (st_i, pos_i, acc_i), else **fail**
- Compute $t(st_i, sym_i) = (st_{i+1}, sym_{i+1}, pos_{i+1})$

  If st_{i+1} is a reject state, output **fail**
  else if st_{i+1} is accept, output **F(*k*,acc)**

  Next Step
- acc_{i+1} = **Update Accumulator**$(pos_i, sym_{i+1}, acc_i)$
- $r' = F(k_A, (acc, t))$; $(SK'_A, VK'_A) = Setup(1^k; r_A')$
  Output **$(st_{i+1}, sym_{i+1}, pos_{i+1}, acc_{i+1})$**
  Output **New Signature on $(st_{i+1}, pos_{i+1}, acc_{i+1})$**

O(C′)

CPRF Construction: Almost Final

$(t, st_i, sig, sym_i, pos_i, acc_i, acc)$

M, T, k, k_A

- **Verify Current Accumulator Value** acc_i, else **fail**
- $r = F(k_A, (acc, t-1))$; $(SK_A, VK_A) = Setup(1^k; r_A)$
  Verify Signature on (st_i, pos_i, acc_i), else **fail**
- Compute $t(st_i, sym_i) = (st_{i+1}, sym_{i+1}, pos_{i+1})$

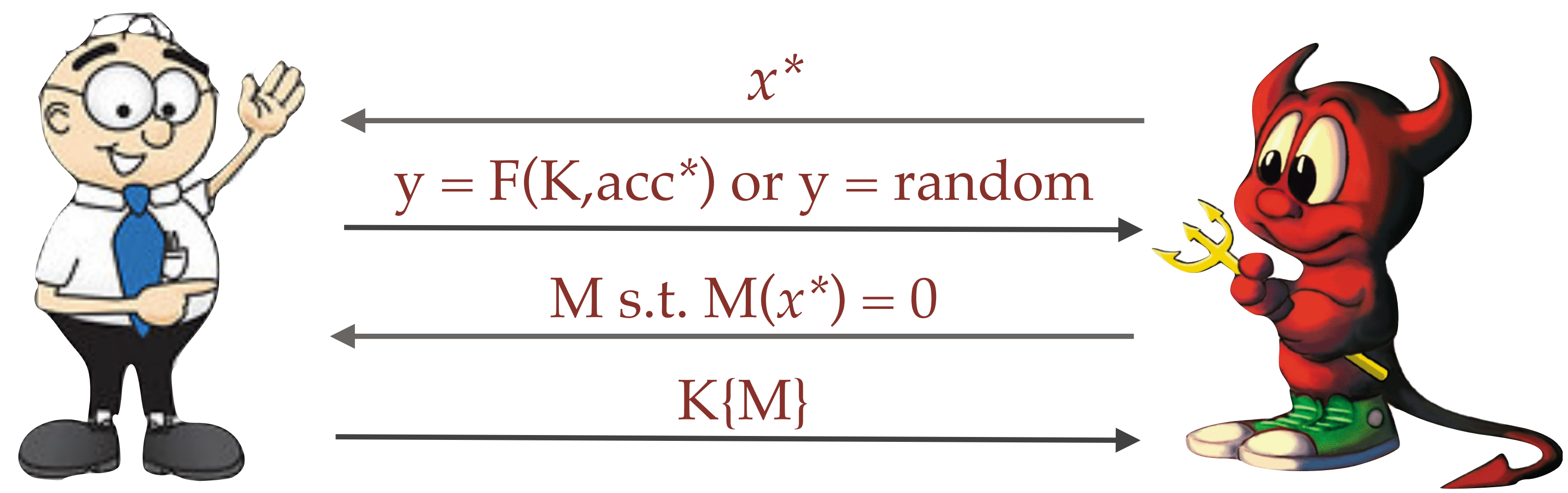
  If st_{i+1} is a reject state, output **fail**
  else if st_{i+1} is accept, output **F(*k*,acc)**

  Next Step
- acc_{i+1} = **Update Accumulator**$(pos_i, sym_{i+1}, acc_i)$
- $r' = F(k_A, (acc, t))$; $(SK'_A, VK'_A) = Setup(1^k; r_A')$
  Output $(st_{i+1}, sym_{i+1}, pos_{i+1}, acc_{i+1})$
  Output **New Signature on** $(st_{i+1}, pos_{i+1}, acc_{i+1})$

O(C')

Proof Intuition

Proof Intuition

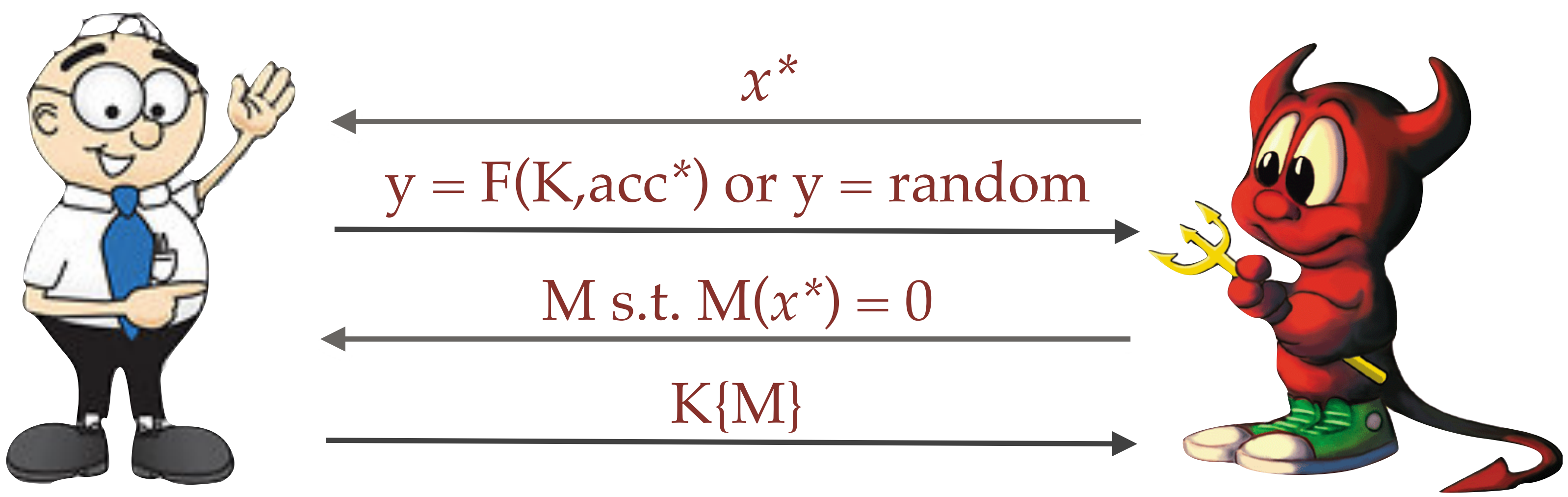


Proof Intuition

x

k, M

If M(x) = 1
output F(k,acc)
else fail



Proof Intuition

x

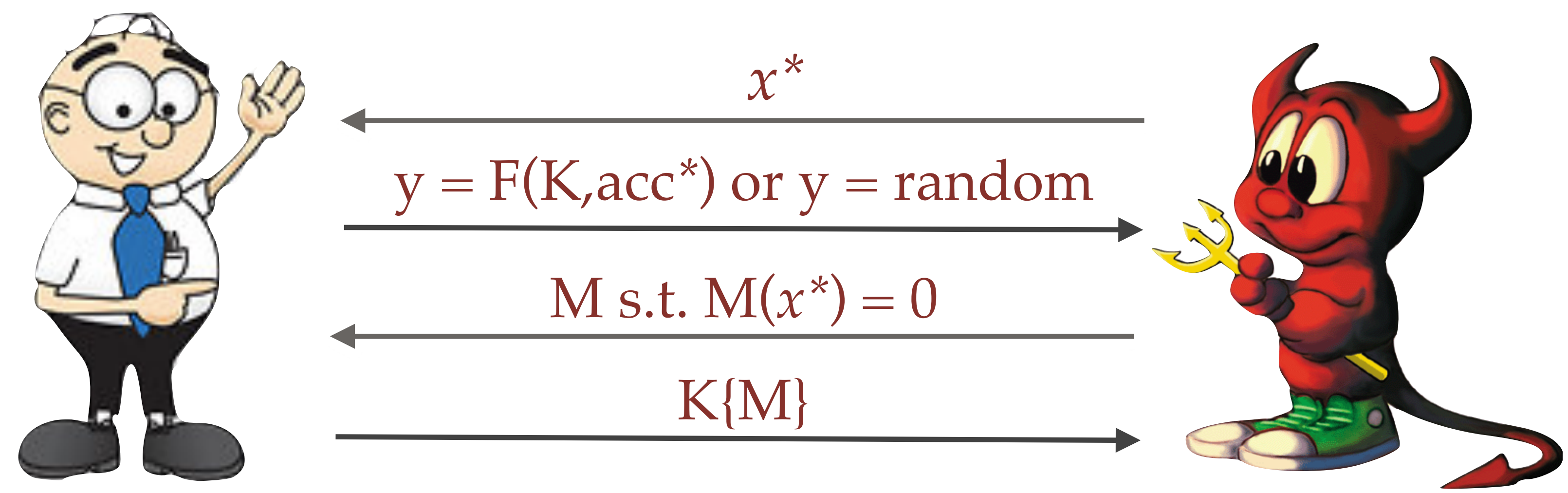
k, M

If M(x) = 1
  output F(*k*,acc)
else fail

x

k{acc*}, M
$y^* = F(K, acc^*)$

If M(x) = 1
  if acc = acc*
    output y*
  else output F(*k*,acc)
else fail

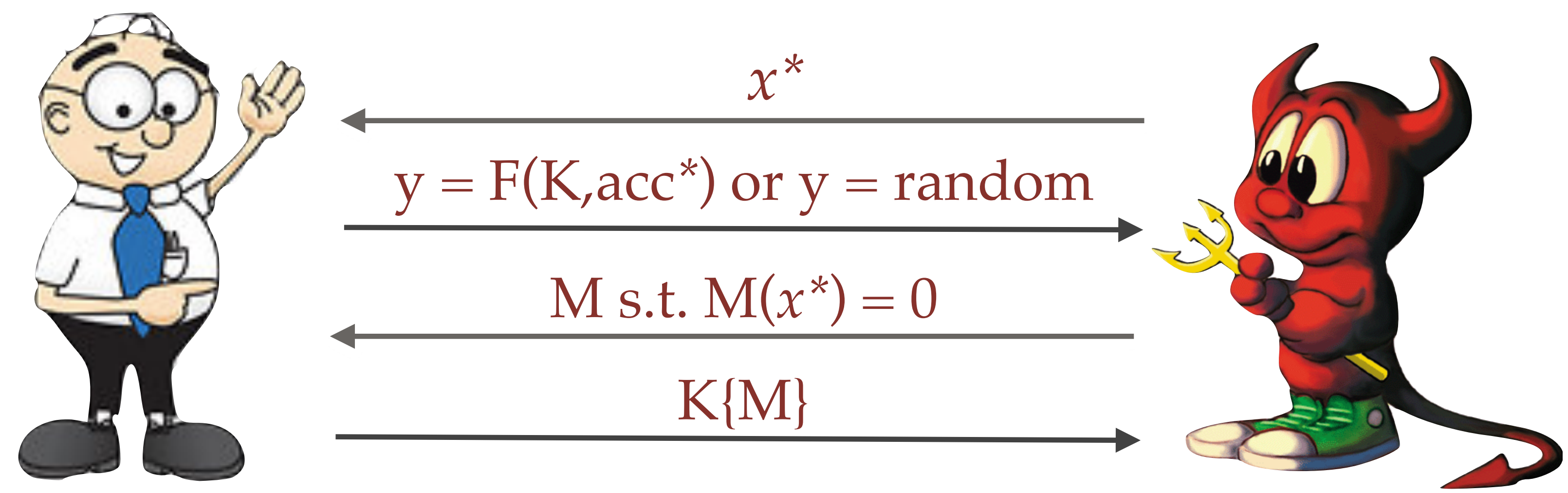


Proof Intuition

x

k, M

If M(x) = 1
  output F(k,acc)
else fail



x

k\{acc*\}, M
$y^* = F(K,\text{acc}^*)$

If M(x) = 1
  if acc = acc*
    output y*
  else output F(k,acc)
else fail

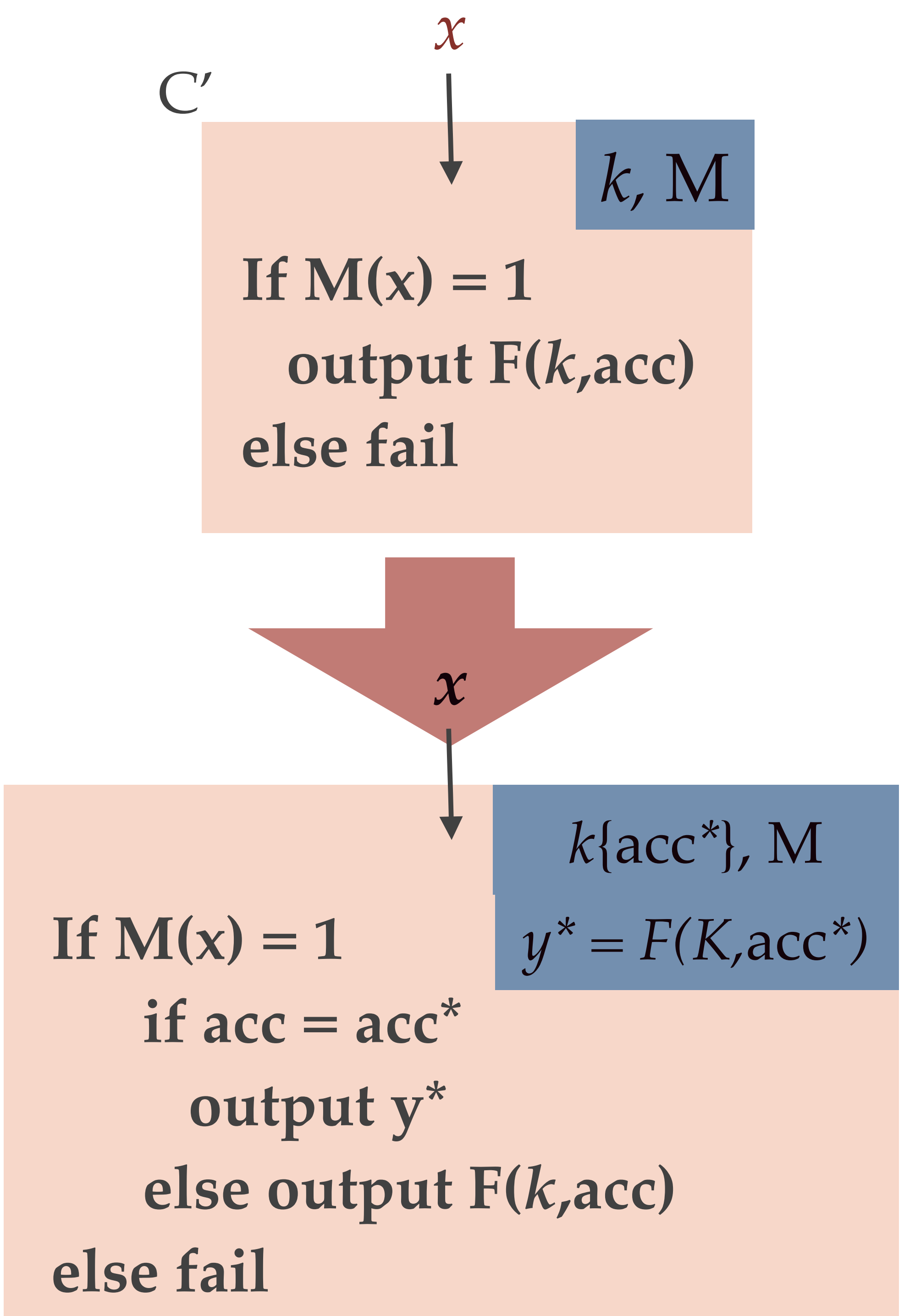
x

k\{acc*\}, M
y*= random

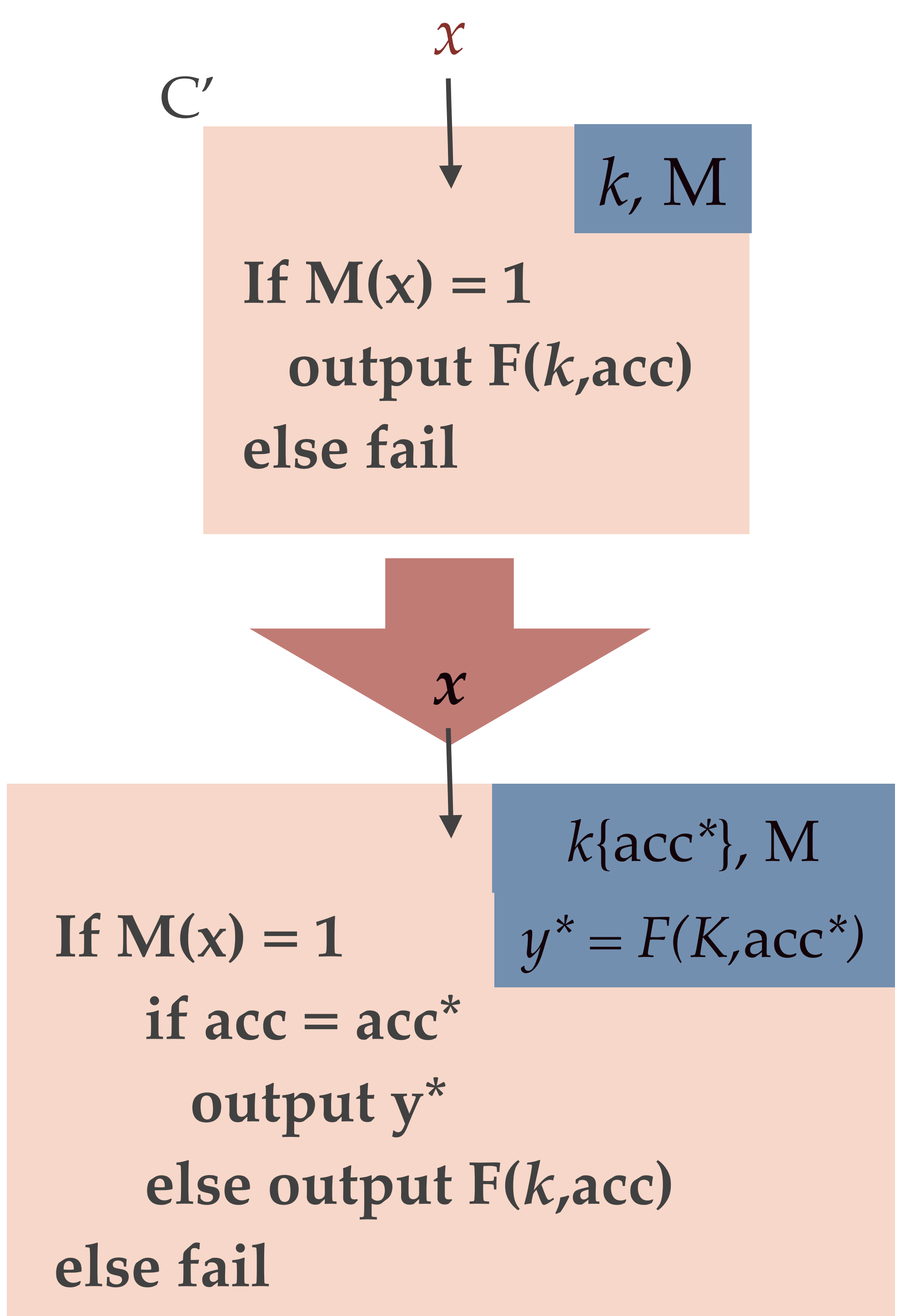
If M(x) = 1
  if x = x*
    output y*
  else output F(k,acc)
else fail

Proof Intuition

Proof Intuition

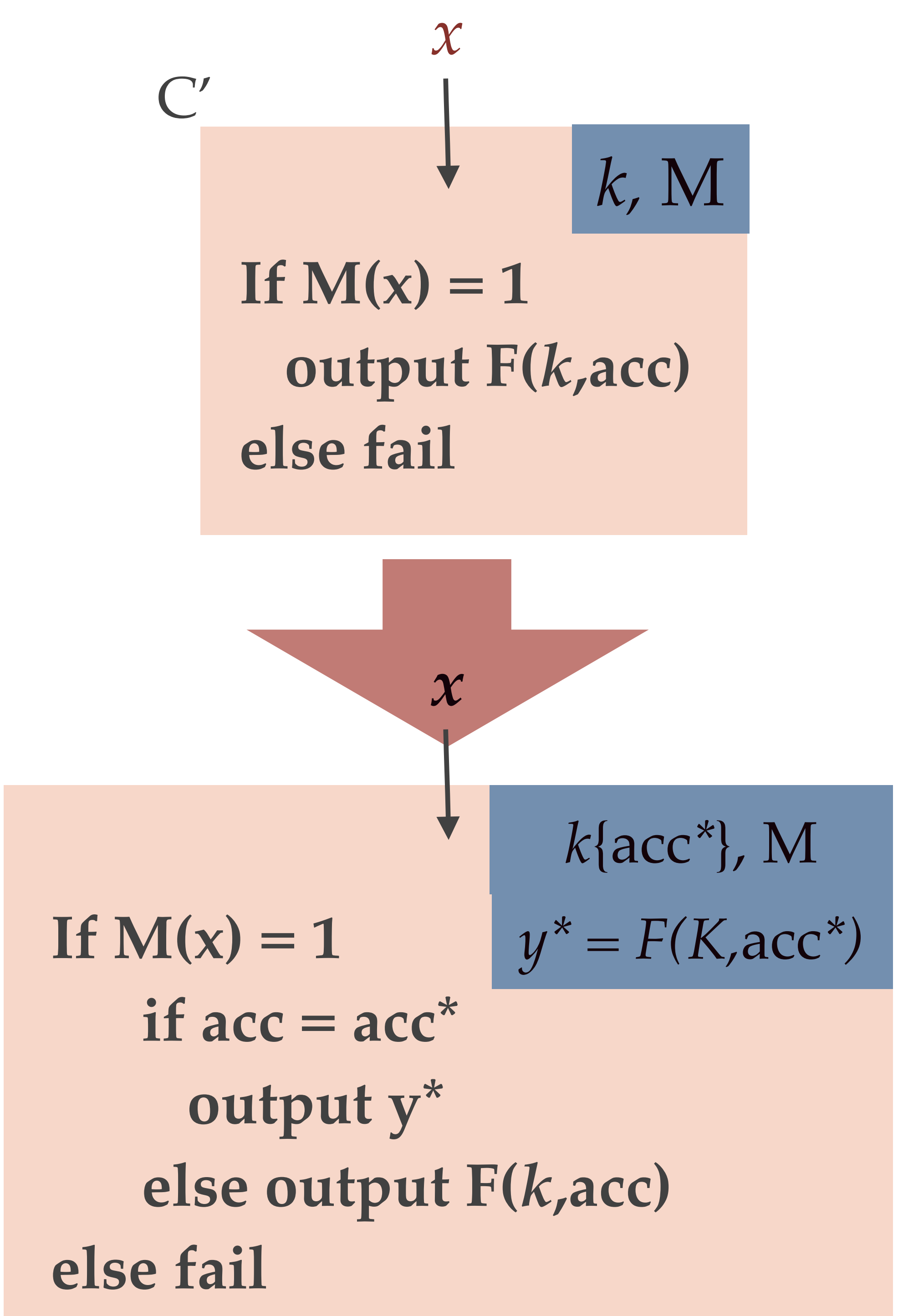


Proof Intuition



Ensure that O(C′) does not reach accept state for any input with acc = acc* :

Proof Intuition

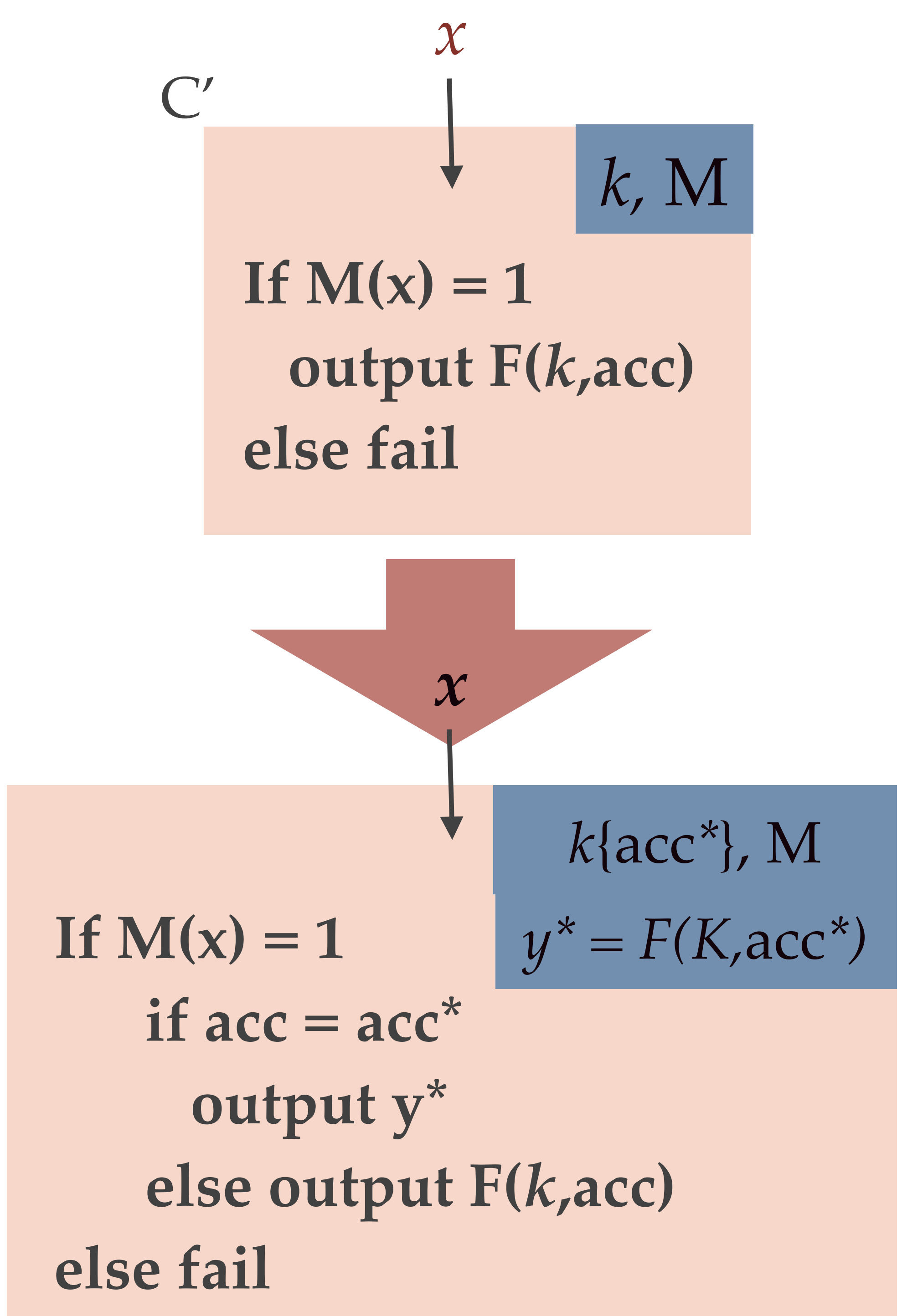


Ensure that O(C′) does not reach accept state for any input with acc = acc* :

Steps taken by M on x*

1. C′ does not reach accept state for any input with acc = acc* within first t* steps

Proof Intuition



Ensure that O(C′) does not reach accept state for any input with acc = acc* :

Steps taken by M on x*

1. C′ does not reach accept state for any input with acc = acc* within first t* steps

2. C′ does not reach accept state for any input with acc = acc* for t > t* steps

Proof Intuition

Proof Intuition

1. C' does not reach accept state for any input with acc = acc* within first t* steps

2. C' does not reach accept state for any input with acc = acc* for $t > t^*$ steps

Proof Intuition

1. C' does not reach accept state for any input with $acc = acc^*$ within first t^* steps

2. C' does not reach accept state for any input with $acc = acc^*$ for $t > t^*$ steps

$O(t^*)$ hybrids using properties of:

- positional accumulator
- splittable signatures

Proof Intuition

1. C′ does not reach accept state for any input with acc = acc* within first t* steps

2. C′ does not reach accept state for any input with acc = acc* for t > t* steps

Cannot use hybrids for each step i > t

Exponential hybrids

Proof Intuition

1. C′ does not reach accept state for any input with acc = acc* within first t* steps

2. C′ does not reach accept state for any input with acc = acc* for $t > t^*$ steps

Novel technique using PRG

Cannot use hybrids for each step $i > t$

Exponential hybrids

Final Construction

$(t, st_i, sig, sym_i, pos_i, acc_i, \underline{seed}, proof, acc_n)$

Final Construction

(t, st_i ,sig, sym_i, pos_i , acc_i, seed, proof, acc_n)

M, T, PP, k, k_1, k_2, ……, k_l

Final Construction

(t, st_i ,sig, sym_i, pos_i , acc_i, seed, proof, acc_n)

M, T, PP, k, k_1, k_2, ……, k_l

- **Verify PRG(seed) =** PRG(s_m) for $2^m <= t < 2^{m+1}$, else **fail**

m^{th} interval

Final Construction

$(t, st_i, sig, sym_i, pos_i, acc_i, \underline{seed}, proof, acc_n)$

M, T, PP, k, k₁, k₂, ……, kₗ

- **Verify PRG(seed) =** PRG(s_m) for $2^m <= t < 2^{m+1}$, else **fail**

m^{th} interval

m^{th} landmark

- If $t+1 = 2^{m+1}$, then **new seed** = s_{m+1} ; else **new seed** = `` ''

Final Construction

(t, st_i ,sig, sym_i, pos_i , acc_i, seed, proof, acc_n)

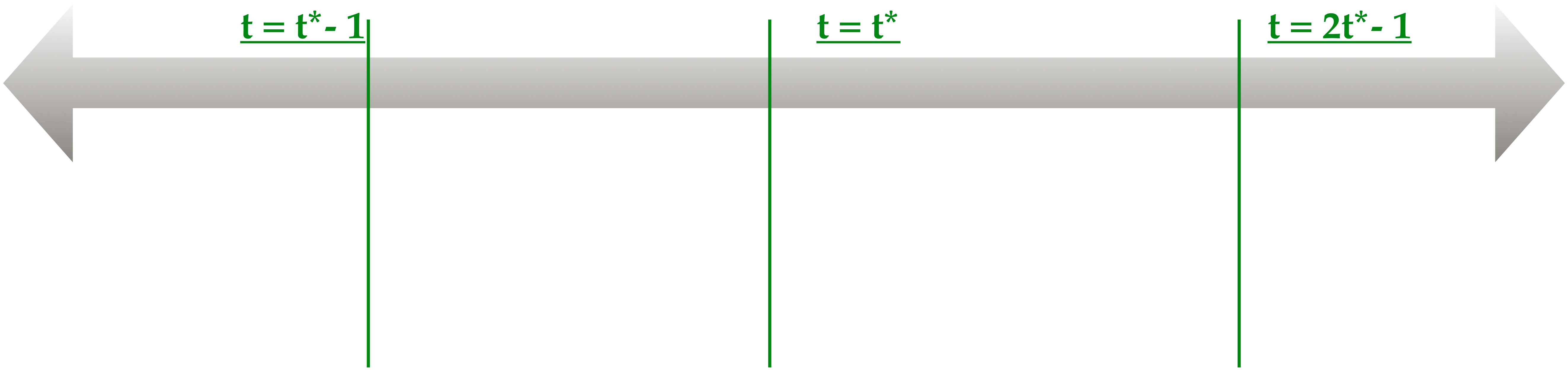
M, T, PP, k, k₁, k₂, ……, k_l

- **Verify PRG(seed) =** PRG(s_m) for $2^m <= t < 2^{m+1}$, else **fail**
- **Verify Current Accumulator Value** acc_i, else **fail**
- **Verify Signature on** (st_i, pos_i , acc_i), else **fail**
- Compute next step
- acc_{i+1} = **Update Accumulator**()
- Output **(st_{i+1} ,sym_{i+1}, pos_{i+1} ,acc_{i+1})**
  Output **New Signature on (st_{i+1} , pos_{i+1} , acc_{i+1})**
- If $t+1 = 2^{m+1}$, then **new seed** = s_{m+1} ; else **new seed** = `` ''

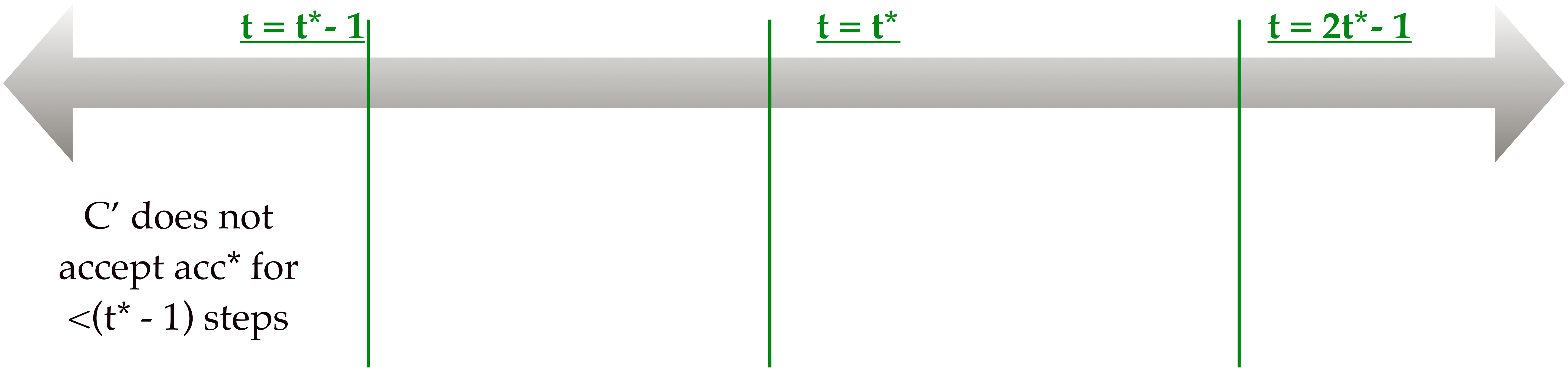
O(C')

Tail Hybrids

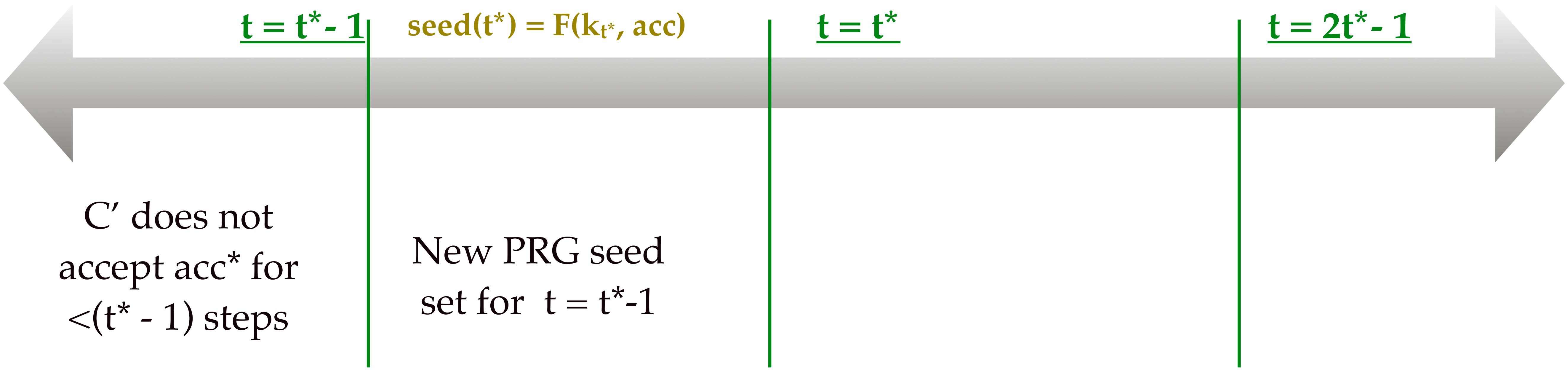
Tail Hybrids



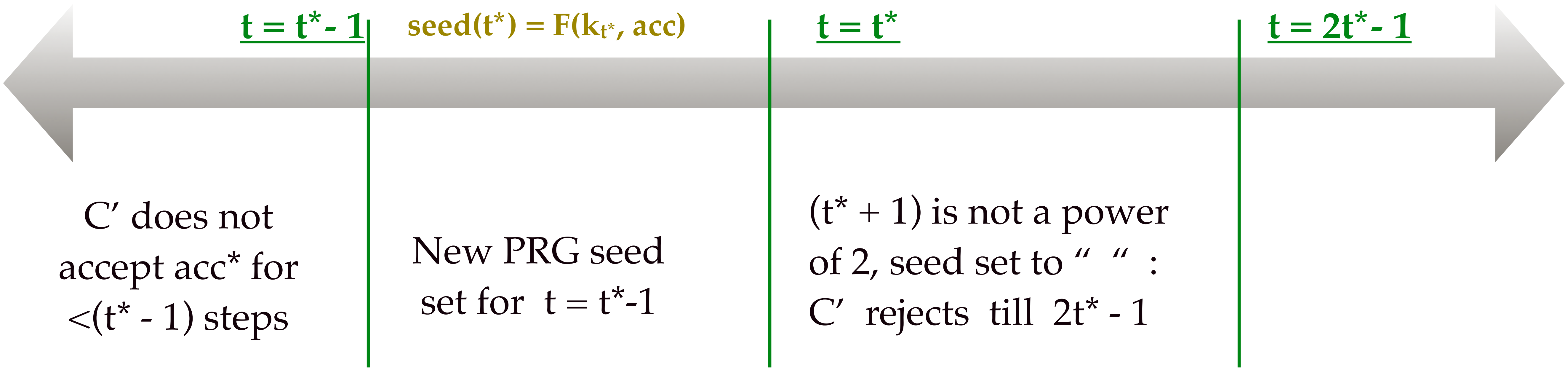
Tail Hybrids



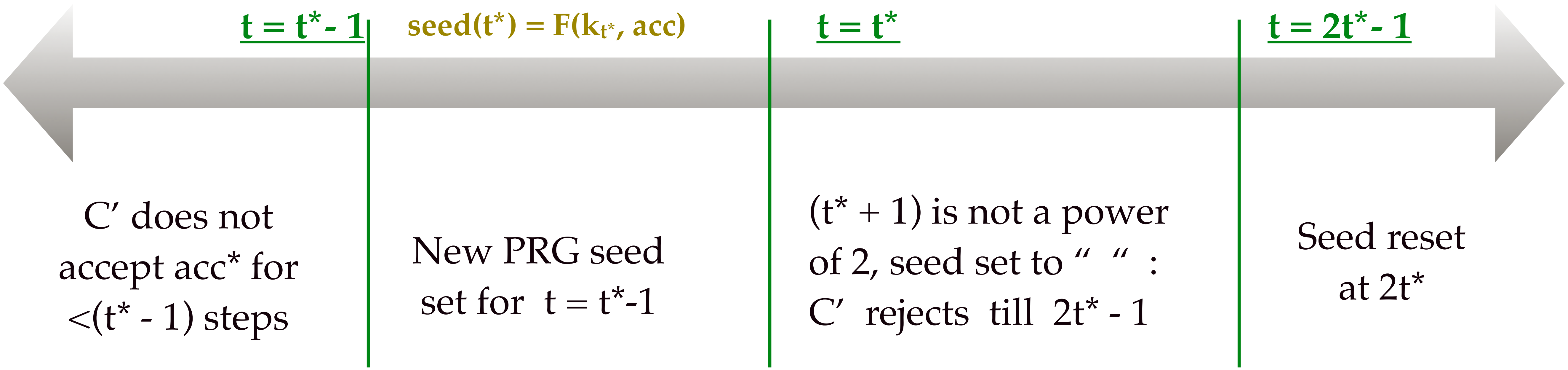
Tail Hybrids



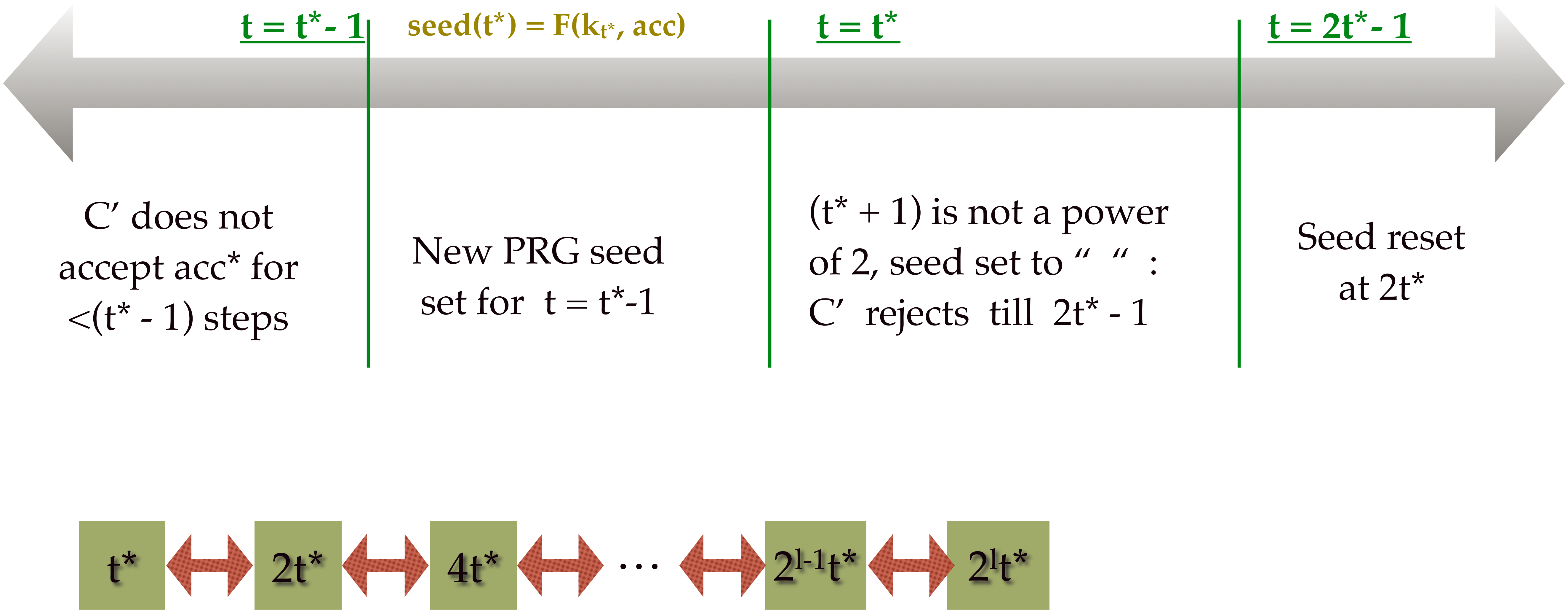
Tail Hybrids



Tail Hybrids



Tail Hybrids



Tail Hybrids

$t = t^* - 1$ | $seed(t^*) = F(k_{t^*}, acc)$ | $t = t^*$ | $t = 2t^* - 1$

C′ does not accept acc* for <(t* - 1) steps

New PRG seed set for t = t*-1

(t* + 1) is not a power of 2, seed set to “ “ : C′ rejects till 2t* - 1

Seed reset at 2t*

$t^* \leftrightarrow 2t^* \leftrightarrow 4t^* \leftrightarrow \cdots \leftrightarrow 2^{l-1}t^* \leftrightarrow 2^{l}t^*$

Possible to have O(k) hybrids for $O(2^k)$ steps !

Tail Hybrids

$t = t^* - 1$ | $seed(t^*) = F(k_{t^*}, acc)$ | $t = t^*$ | $t = 2t^* - 1$

C′ does not accept acc* for <(t* - 1) steps

New PRG seed set for $t = t^*-1$

$(t^* + 1)$ is not a power of 2, seed set to “ “ : C′ rejects till $2t^* - 1$

Seed reset at $2t^*$

t^* ⟷ $2t^*$ ⟷ $4t^*$ ⟷ … ⟷ $2^{l-1}t^*$ ⟷ $2^{l}t^*$

Possible to have $O(k)$ hybrids for $O(2^k)$ steps !

Conclusion

Construction of Constrained PRF scheme for unbounded inputs

Conclusion

Construction of Constrained PRF scheme for unbounded inputs

- **Using ``*iO-friendly*" primitives of positional accumulator, splittable signatures**
- **Using a novel technique for reducing number of hybrids in the proof**

Construction of Attribute-based Encryption for Turing machines

Conclusion

Construction of Constrained PRF scheme for unbounded inputs

- **Using ``*iO-friendly*" primitives of positional accumulator, splittable signatures**
- **Using a novel technique for reducing number of hybrids in the proof**

Construction of Attribute-based Encryption for Turing machines

References

- [AFP'14] Abusalah, H., Fuchsbauer, G., Pietrzak, K.: Constrained prfs for unbounded in- puts. IACR Cryptology ePrint Archive 2014, 840 (2014), http://eprint.iacr. org/2014/840

- [GGHRSW'13] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate in- distinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013)

- [GGHW'14] Garg, S., Gentry, C., Halevi, S., Wichs, D.: On the implausibility of differing- inputs obfuscation and extractable witness encryption with auxiliary input. In: Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. pp. 518–535 (2014)

- [GGM'84] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: FOCS. pp. 464–479 (1984)

- [BW'13] Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: ASIACRYPT. pp. 280–300 (2013)

- [BGI'14] Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: PKC. pp. 501–519 (2014)

References

- [BdM93] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27,1993, Proceedings, pages 274–285, 1993.
- [ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differinginputsobfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. http://eprint.iacr.org/.
- [BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan,and Ke Yang. On the (im)possibility of obfuscating programs. J. ACM, 59(2):6, 2012.
- [SW'14] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC. pp. 475–484 (2014)
- [KPTZ'13] Kiayias,A.,Papadopoulos,S.,Triandopoulos,N.,Zacharias,T.:Delegatable pseudorandom functions and applications. In: ACM Conference on Computer and Communications Security. pp. 669–684 (2013)