Laconic Branching Programs from the Diffie-Hellman Assumption

Sanjam Garg, Mohammad Hajiabadi, Peihan Miao, and Alice Murphy

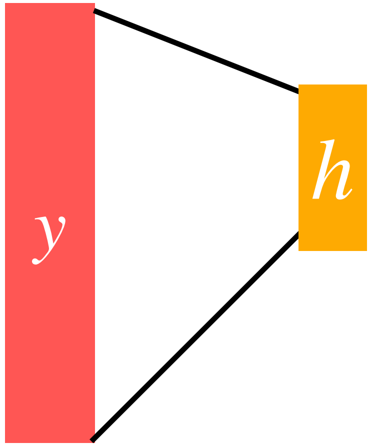
PKC 2024

eprint.iacr.org/2024/102

Laconic Cryptography [CDG+17]

Receiver

Input:

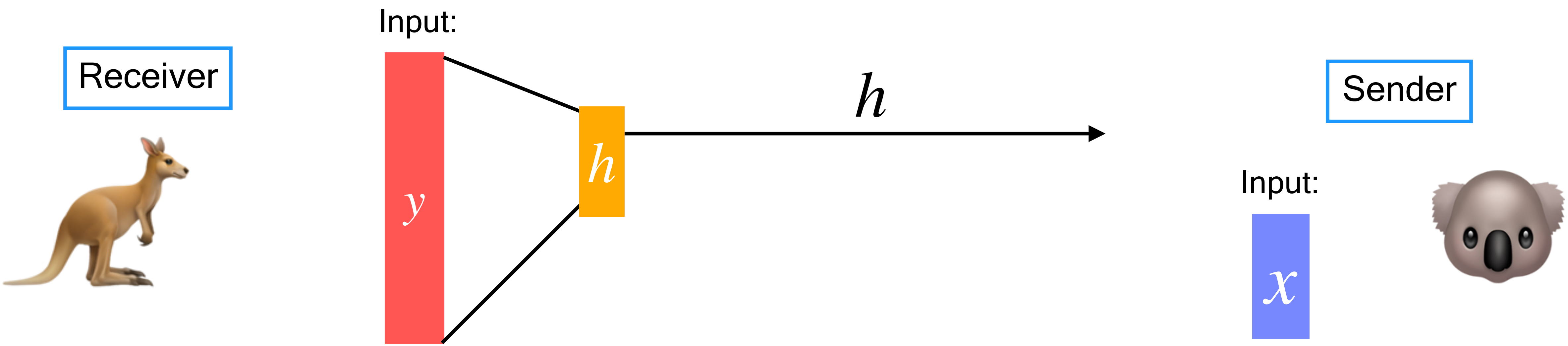


Sender

Input:

$|y| \gg |x|$

Laconic Cryptography [CDG+17]



Receiver

Input:

y

h

h

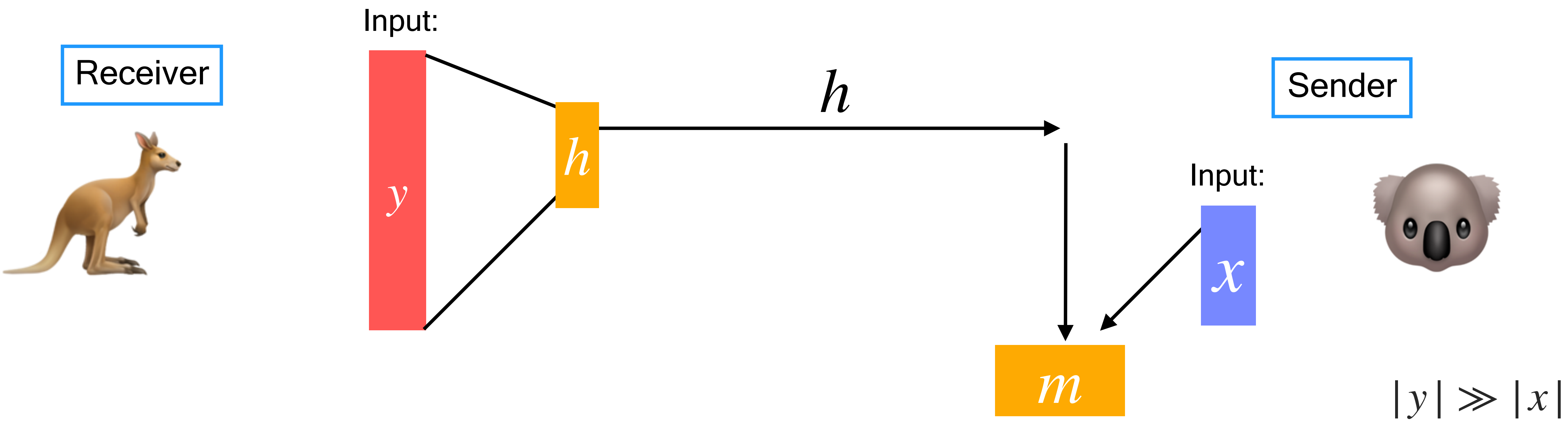
Sender

Input:

x

$|y| \gg |x|$

Laconic Cryptography [CDG+17]



Receiver

Input:

y

h

h

m

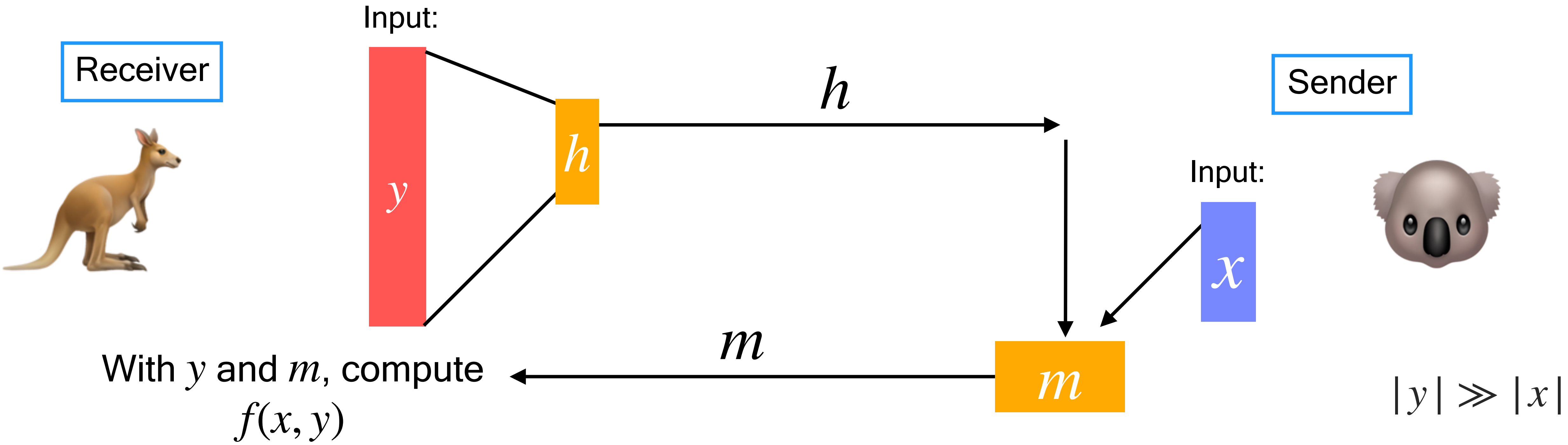
Sender

Input:

x

$|y| \gg |x|$

Laconic Cryptography [CDG+17]



Receiver

Input:

y

h

h

Sender

Input:

x

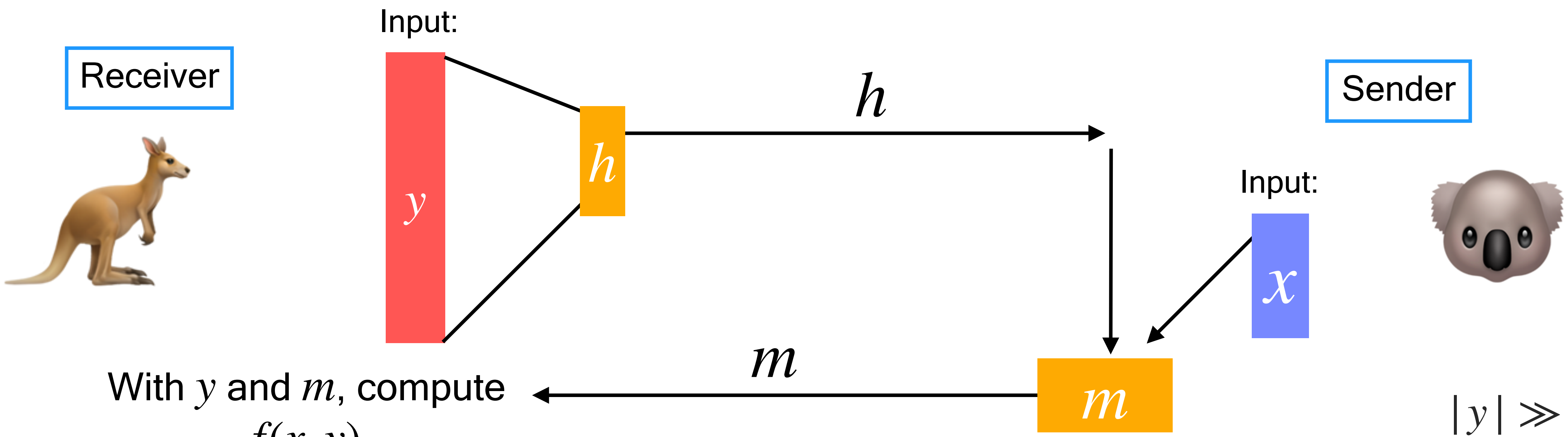
m

m

With y and m, compute
$f(x, y)$

$|y| \gg |x|$

Laconic Cryptography [CDG+17]



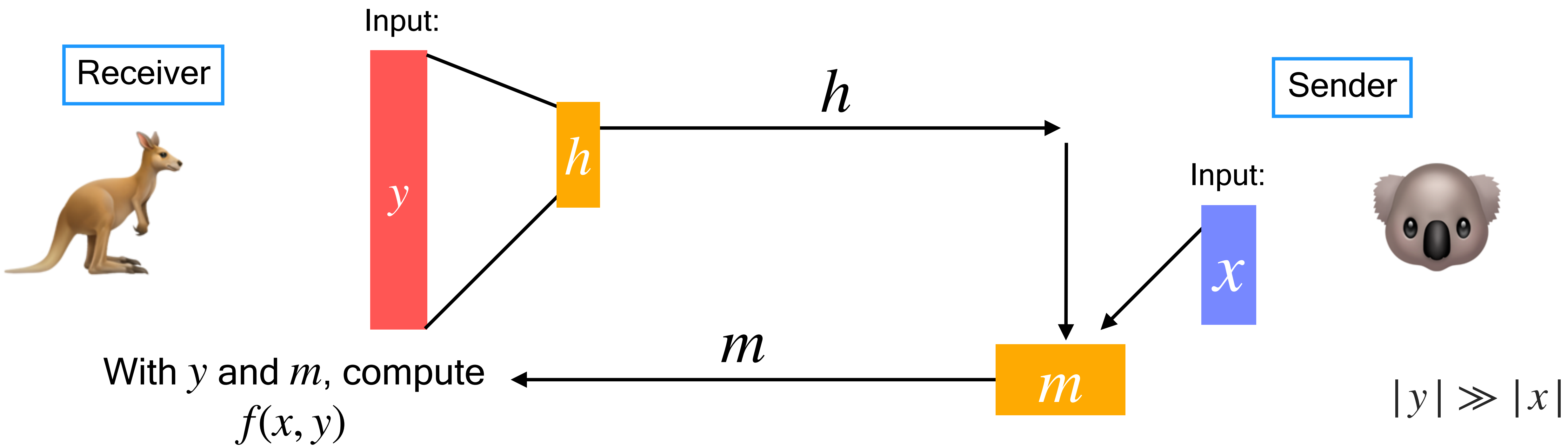
With y and m, compute
$f(x, y)$

$|y| \gg |x|$

Security:

- h hides y
- m hides x to the extent that $f(x, y)$ hides x

Laconic Cryptography [CDG+17]



Receiver

Input:

y

h

h

Sender

Input:

x

m

m

With y and m, compute $f(x, y)$

$|y| \gg |x|$

- Only 1 round of communication allowed (2 messages)
- Communication complexity does not depend on $|y|$

Security:

- h hides y
- m hides x to the extent that $f(x, y)$ hides x

Comparison to previous work in laconic cryptography

More general

- [ABD+21] gives a protocol for Laconic private set intersection (PSI) of sets based on LWE or CDH

which can be used to get PSI

- [This work] gives a protocol for laconic branching programs based on LWE or CDH

Comparison to previous work in laconic cryptography

More general

- [ABD+21] gives a protocol for Laconic private set intersection (PSI) of sets based on LWE or CDH

which can be used to get PSI

- [This work] gives a protocol for laconic branching programs based on LWE or CDH
- [QWW18] gives a laconic protocol for general functionalities based on LWE

Comparison to previous work in laconic cryptography

More general

- [ABD+21] gives a protocol for Laconic private set intersection (PSI) of sets based on LWE or CDH
- [This work] gives a protocol for laconic branching programs based on LWE or CDH

which can be used to get PSI

- [QWW18] gives a laconic protocol for general functionalities based on LWE

Comparison to previous work in laconic cryptography

More general

- [ABD+21] gives a protocol for Laconic private set intersection (PSI) of sets based on LWE or CDH

which can be used to get PSI

- [This work] gives a protocol for laconic branching programs based on LWE or CDH
- [QWW18] gives a laconic protocol for general functionalities based on LWE

There are also constructions for...

Comparison to previous work in laconic cryptography

More general

- [ABD+21] gives a protocol for Laconic private set intersection (PSI) of sets based on LWE or CDH

which can be used to get PSI

- [This work] gives a protocol for laconic branching programs based on LWE or CDH
- [QWW18] gives a laconic protocol for general functionalities based on LWE

There are also constructions for...

- laconic PSI from pairings [ALOS22]

Comparison to previous work in laconic cryptography

More general

- [ABD+21] gives a protocol for Laconic private set intersection (PSI) of sets based on LWE or CDH

which can be used to get PSI

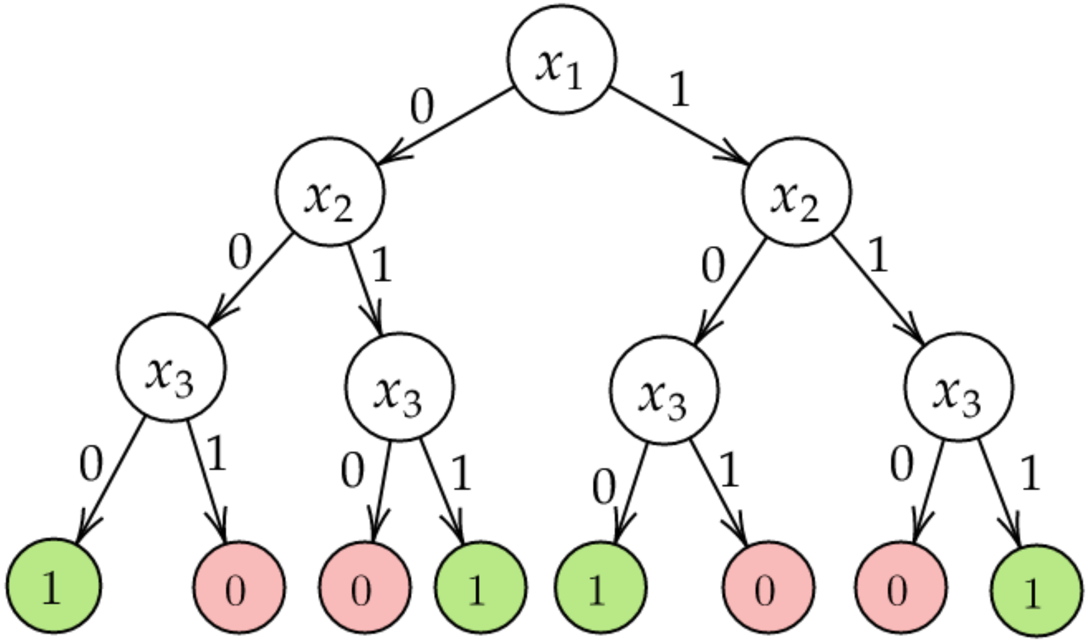
- [This work] gives a protocol for laconic branching programs based on LWE or CDH
- [QWW18] gives a laconic protocol for general functionalities based on LWE

There are also constructions for...

- laconic PSI from pairings [ALOS22]
- laconic oblivious transfer from DDH, CDH, or QR [CDG+17, DG17]

Branching Programs

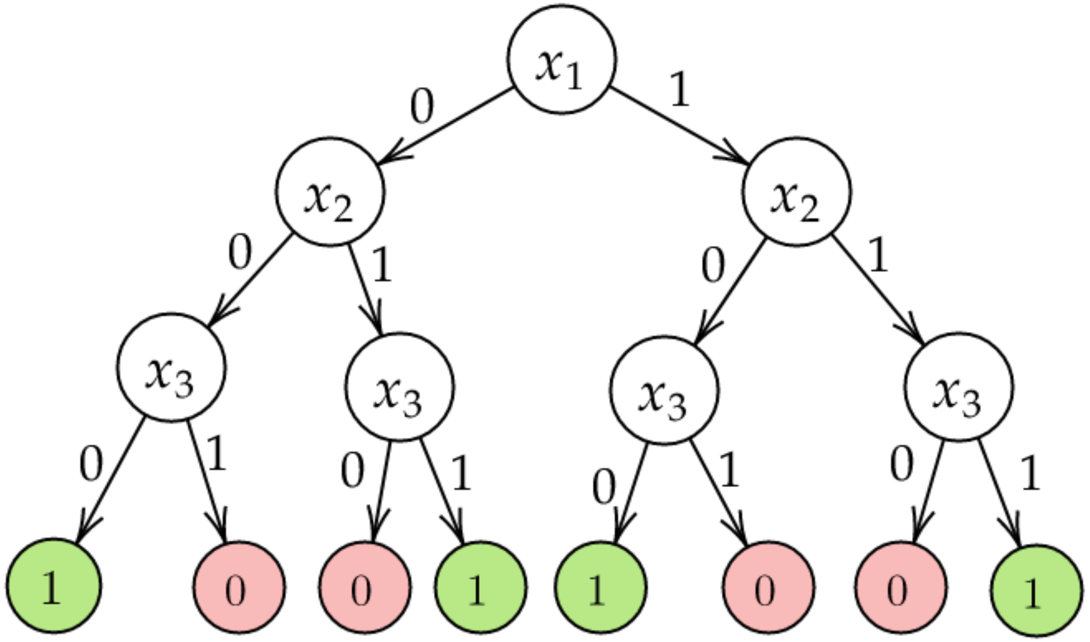
bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

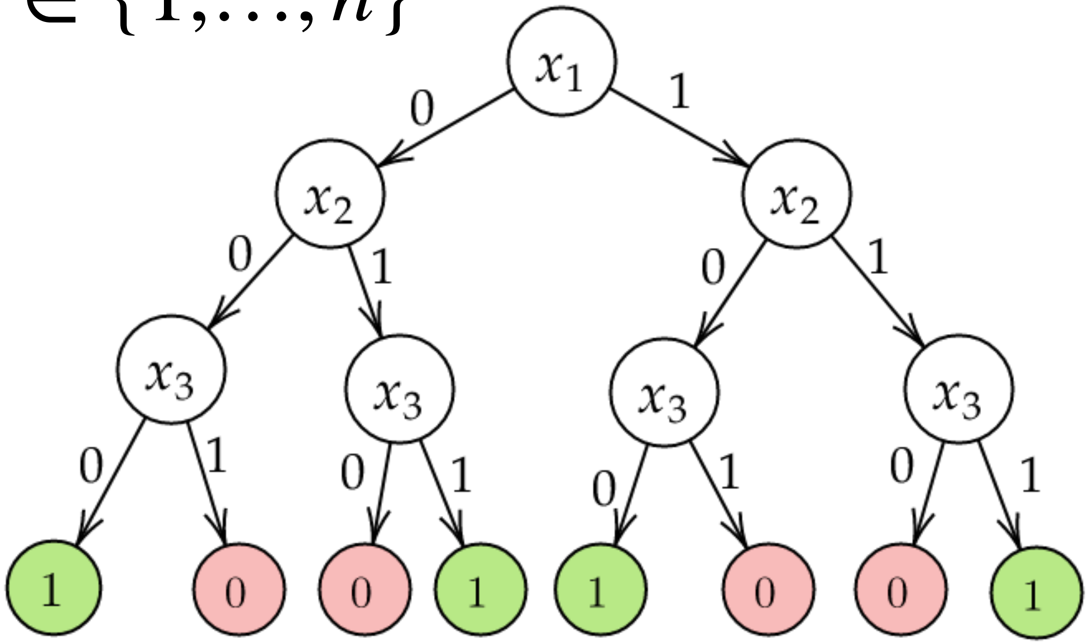
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

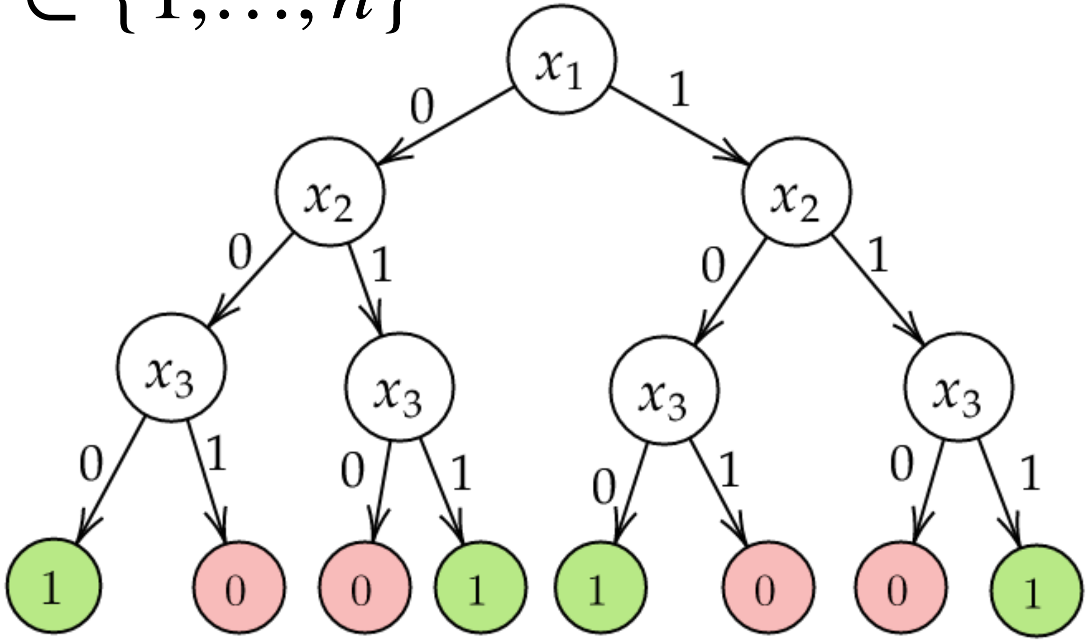
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

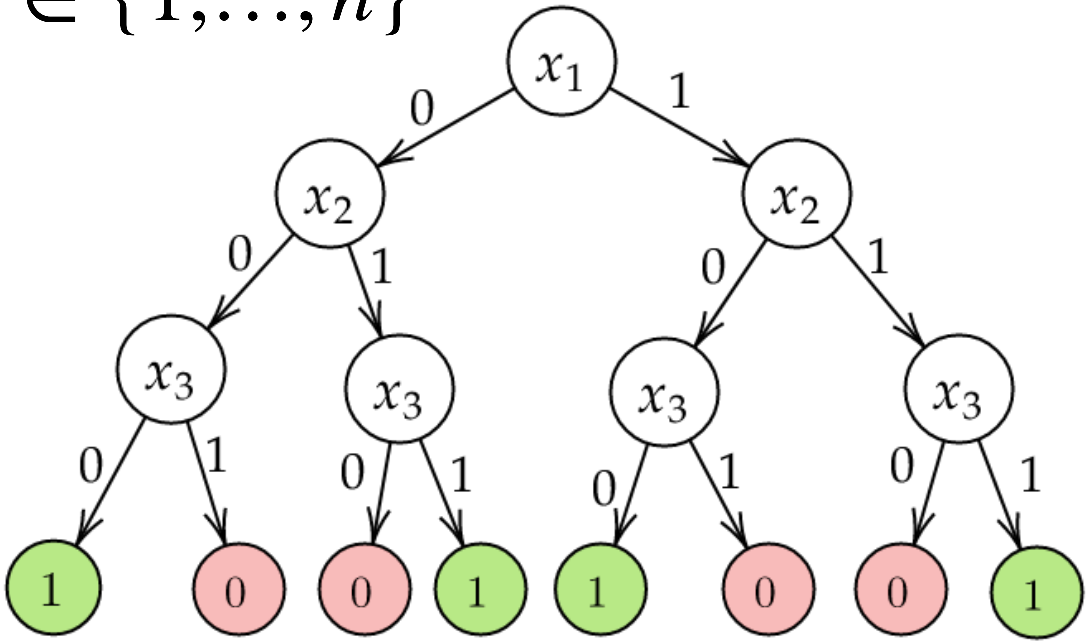
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

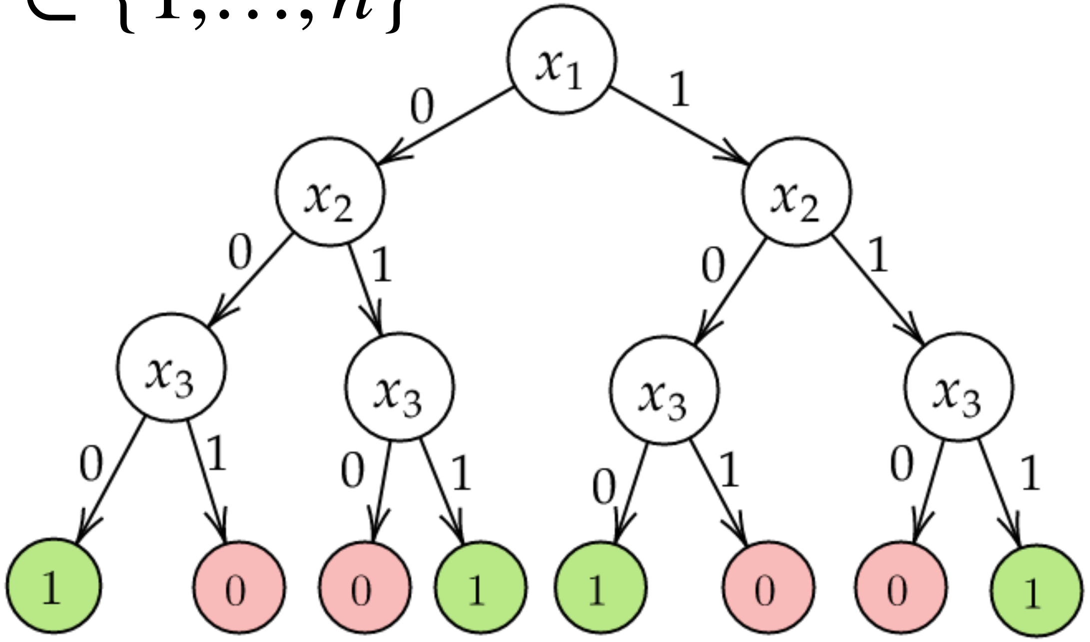
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
    - If $x_i = 0$ then go to the left child



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

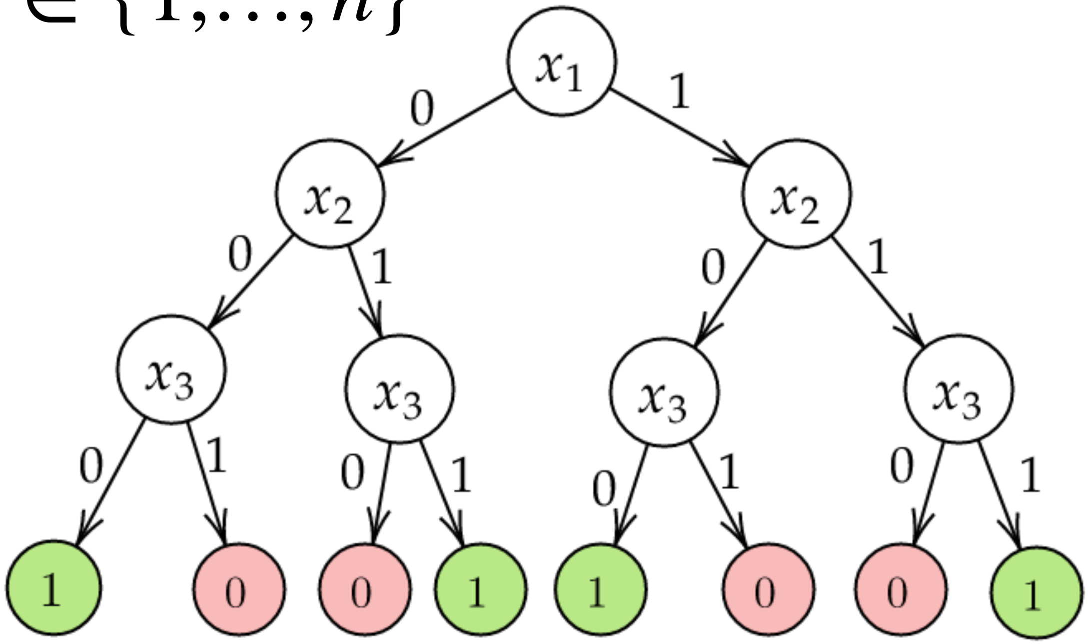
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

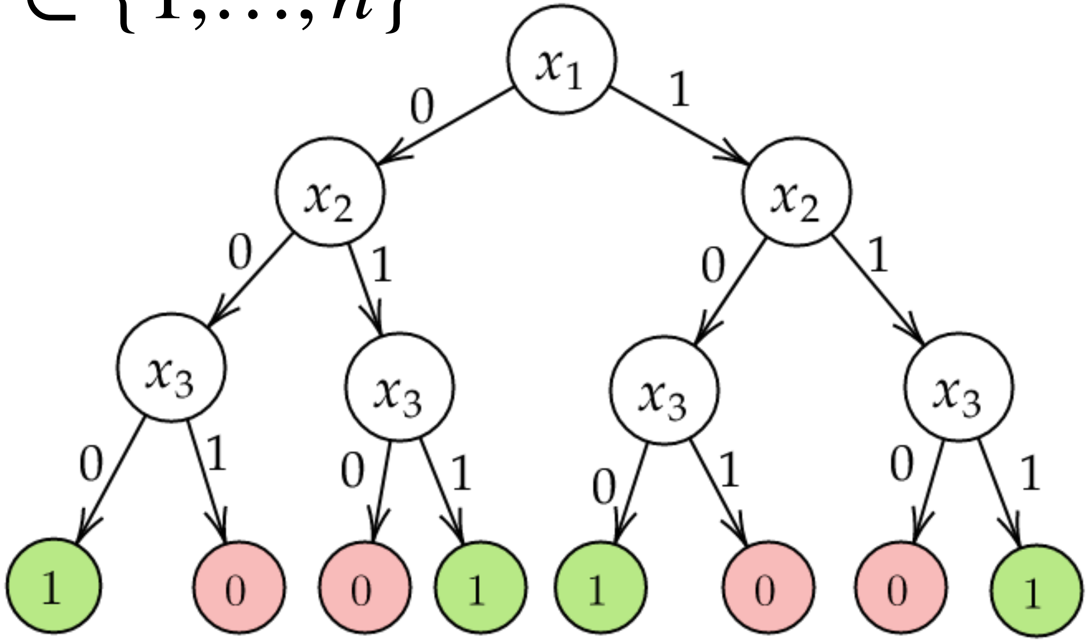
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child
- Each leaf node encodes either



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

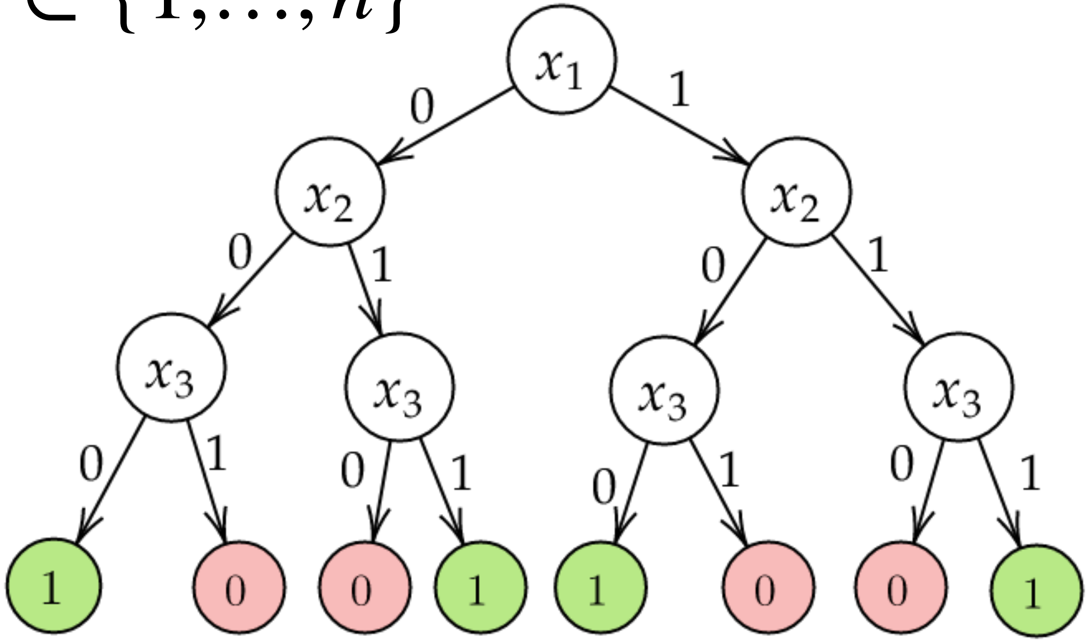
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child
- Each leaf node encodes either
  - 1 (accept) or



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

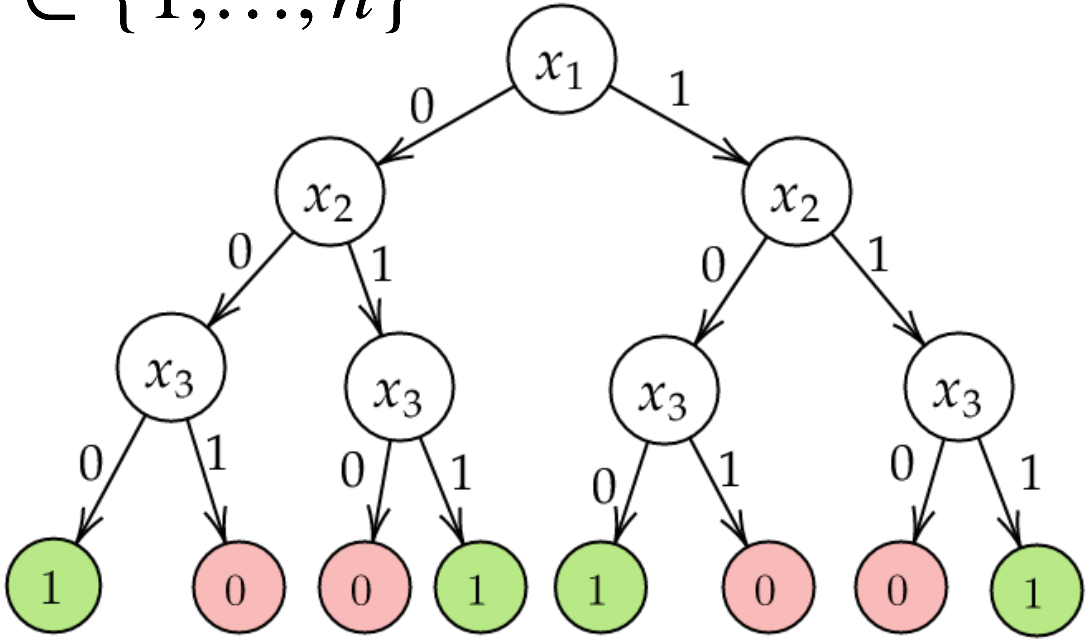
- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child
- Each leaf node encodes either
  - 1 (accept) or
  - 0 (reject)



Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child
- Each leaf node encodes either
  - 1 (accept) or
  - 0 (reject)



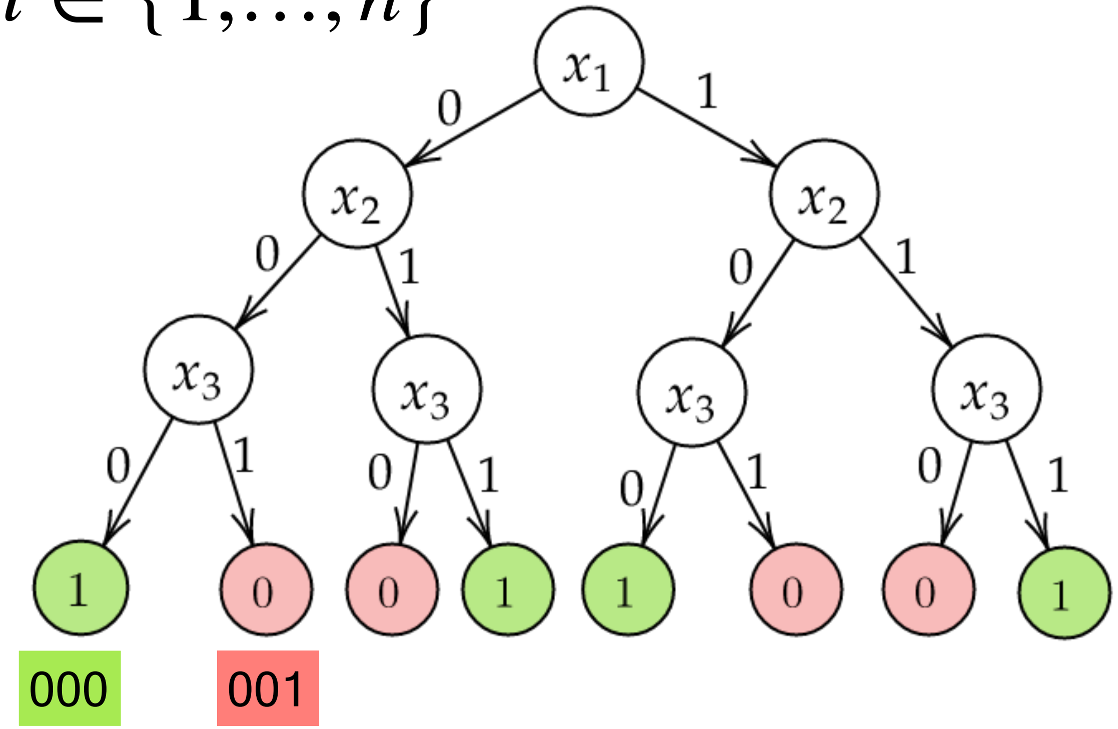
000

BP(000) = 1

Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child
- Each leaf node encodes either
  - 1 (accept) or
  - 0 (reject)

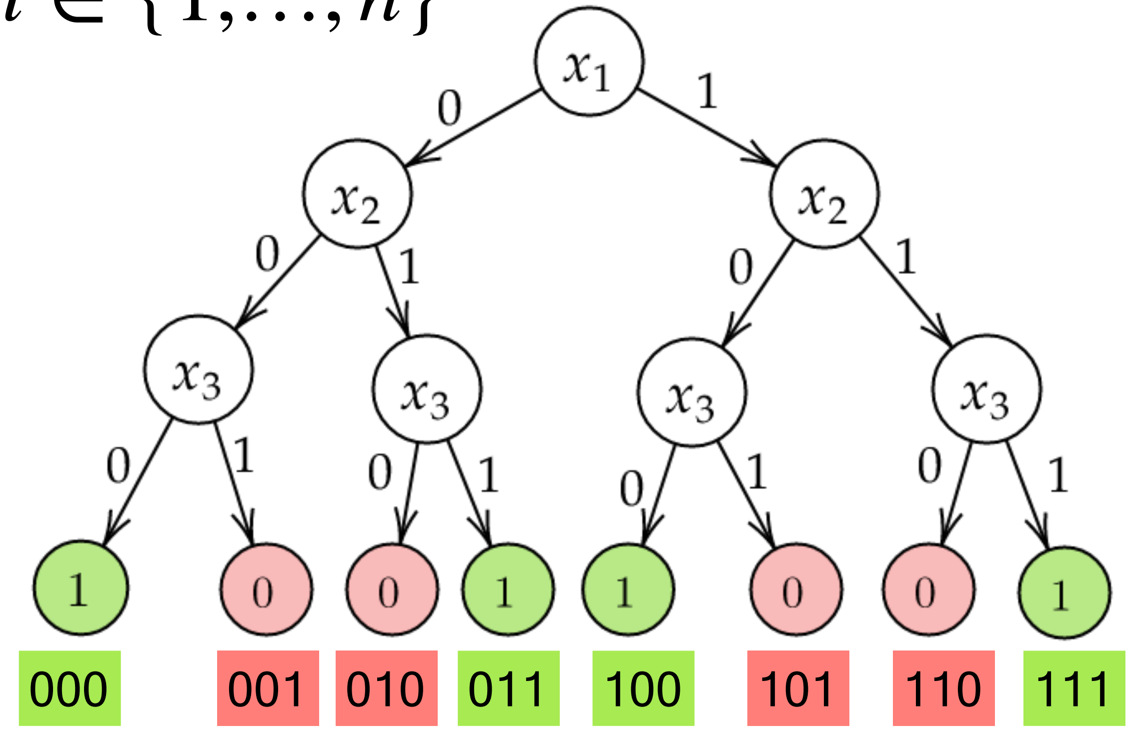


BP(000) = 1 BP(001) = 0

Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child
- Each leaf node encodes either
  - 1 (accept) or
  - 0 (reject)



BP(000) = 1 BP(001) = 0

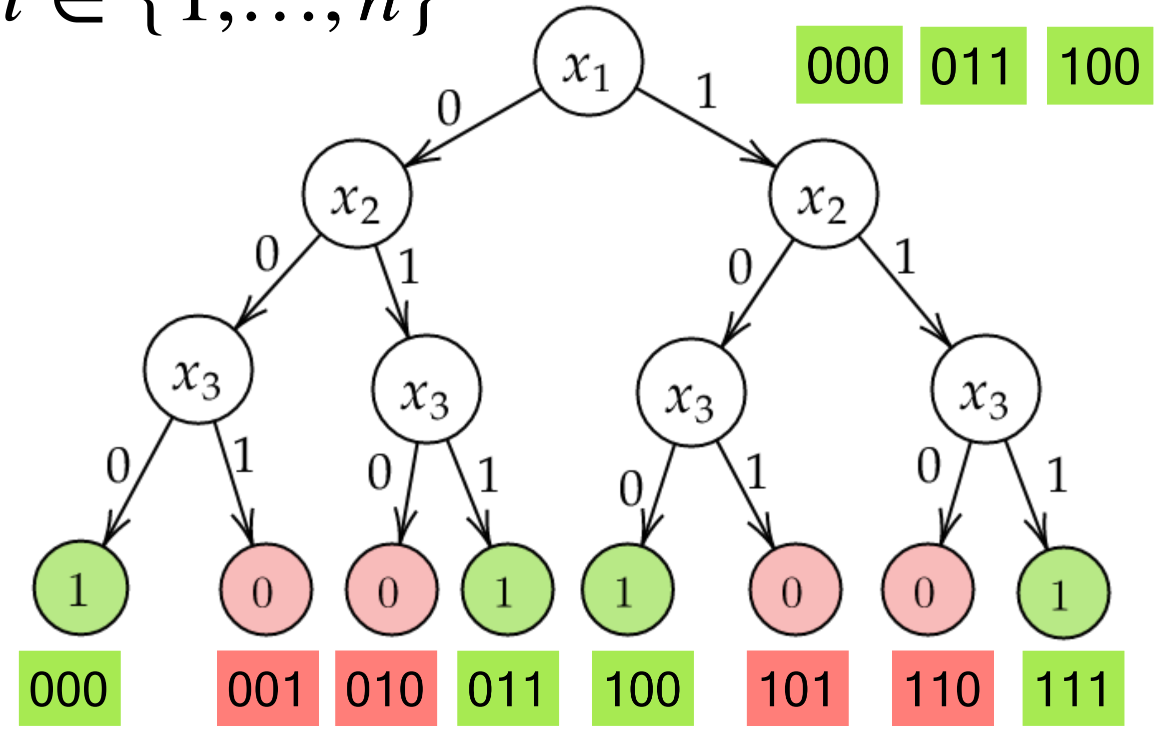
Branching Programs

bit-checking branching programs (for the presentation) but our protocol generalizes to more complicated predicates

- Directed tree that can be evaluated on inputs $x \in \{0,1\}^n$
- Each internal node encodes an index $i \in \{1,\ldots,n\}$
- Evaluate BP(x) starting at the root
  - If $x_i = 0$ then go to the left child
  - If $x_i = 1$ then go to the right child
- Each leaf node encodes either
  - 1 (accept) or
  - 0 (reject)

This BP describes the set:

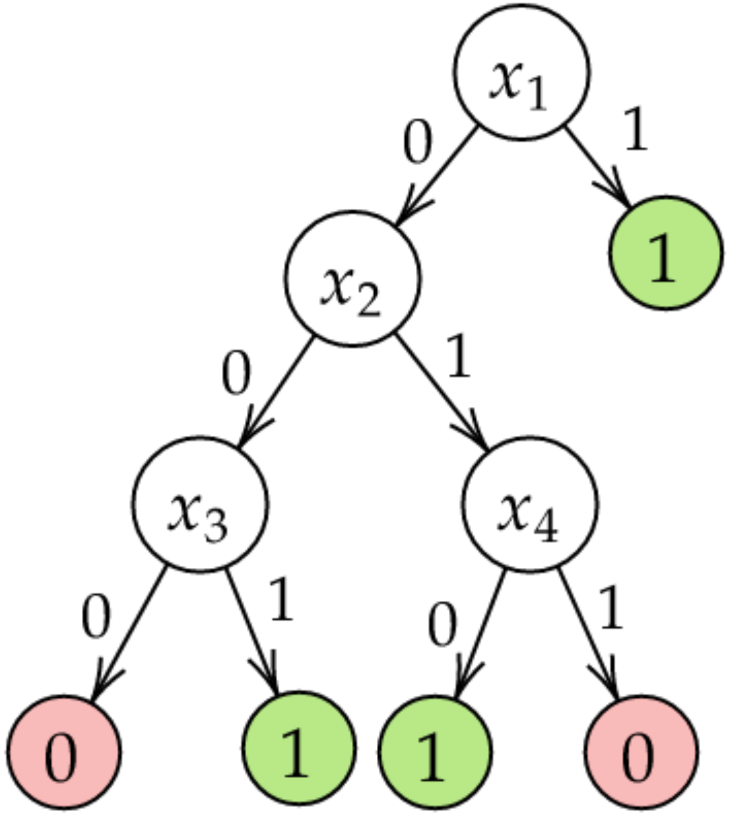
000 011 100 111



BP(000) = 1 BP(001) = 0

Branching Programs — Example 2

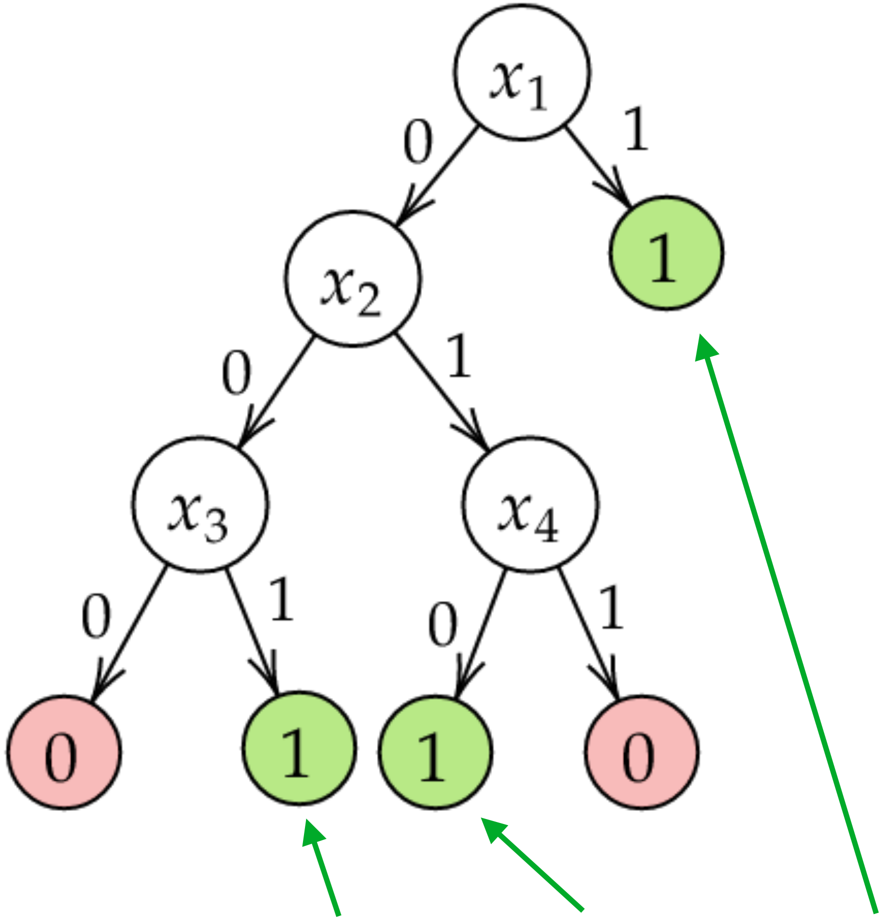
Inputs in $\{0,1\}^4$



- This BP describes the set: $\{001*, 01*0, 1***\}$.

Branching Programs — Example 2

Inputs in $\{0,1\}^4$



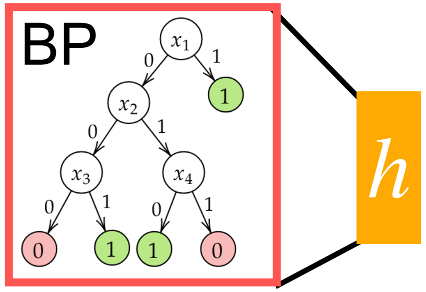
Wildcards

- This BP describes the set: $\{001*, 01*0, 1***\}$.

Laconic Branching Programs

Receiver

Input:



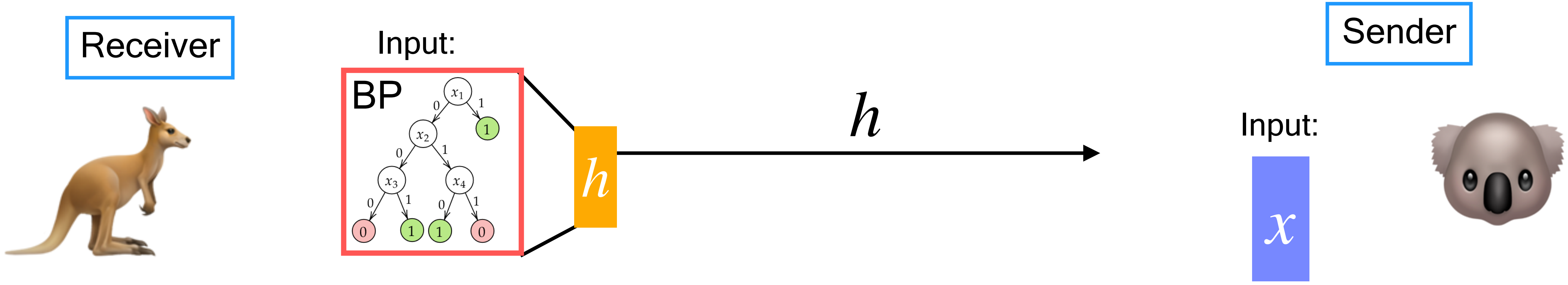
Sender

Input:

x

$|\text{BP}| \gg |x|$

Laconic Branching Programs



Receiver

Input:

h

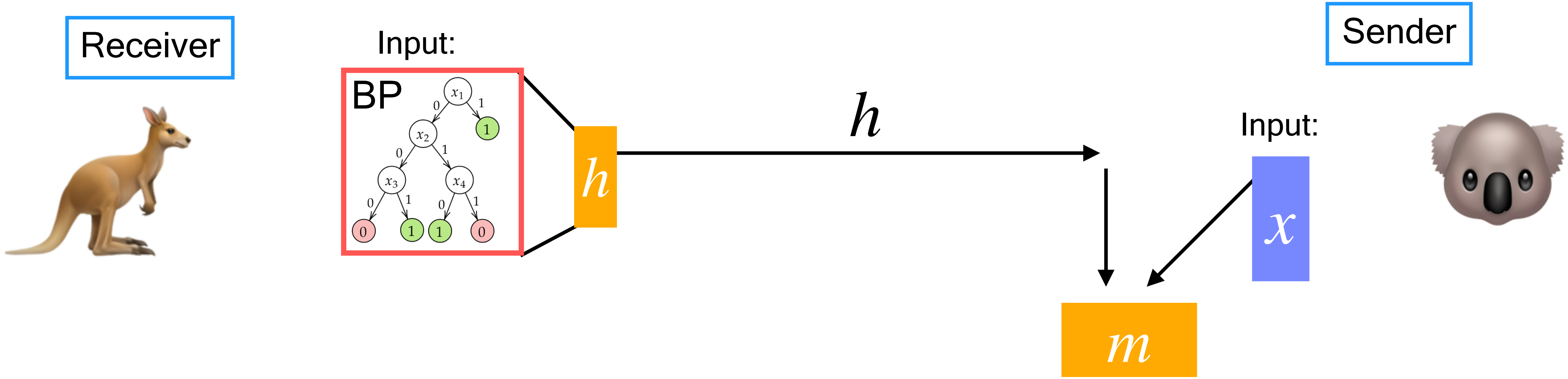
Sender

Input:

x

$|\text{BP}| \gg |x|$

Laconic Branching Programs



Receiver

Input:

BP

h

h

Sender

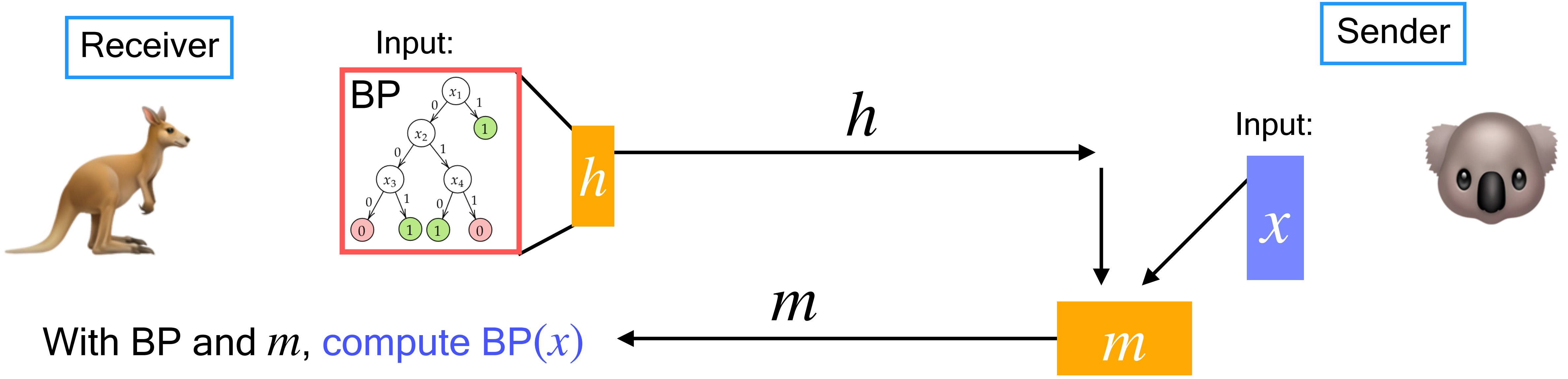
Input:

x

m

$|\text{BP}| \gg |x|$

Laconic Branching Programs



Receiver

Input:

BP

h

h

Sender

Input:

x

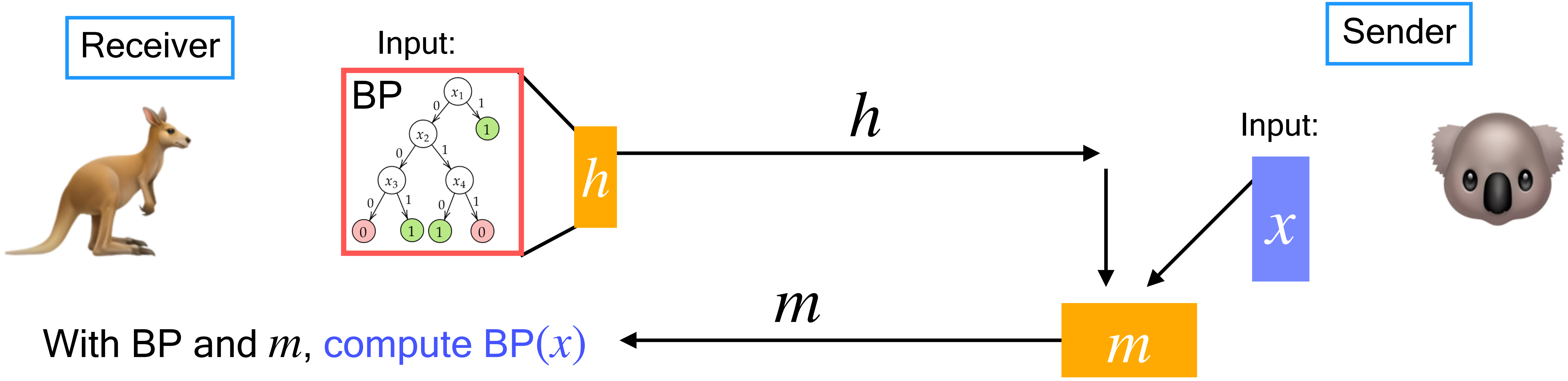
m

m

With BP and m, compute BP(x)

$|\text{BP}| \gg |x|$

Laconic Branching Programs

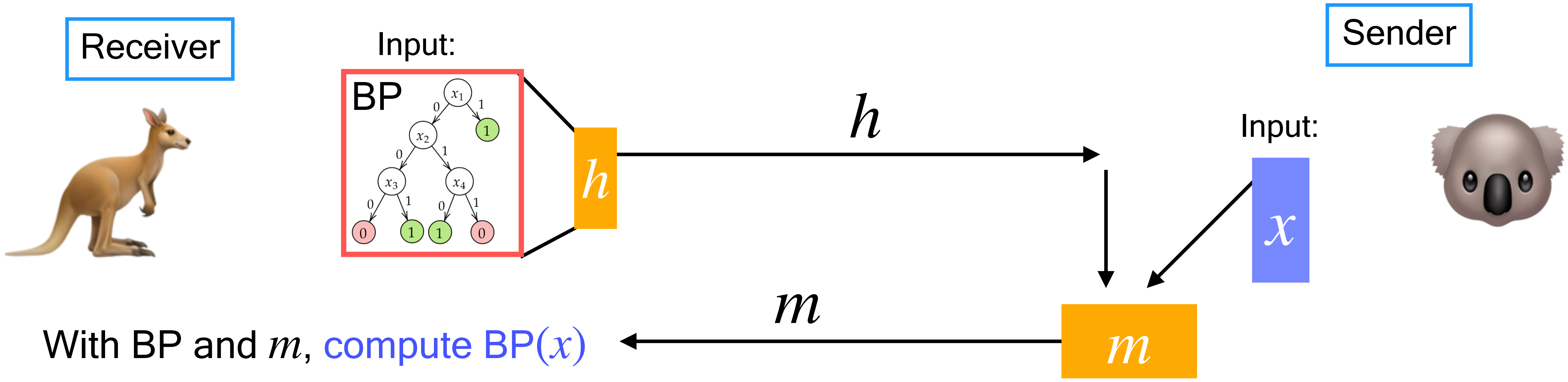


With BP and m, compute BP(x)

$|\text{BP}| \gg |x|$

- Only 1 round of communication allowed (2 messages)

Laconic Branching Programs

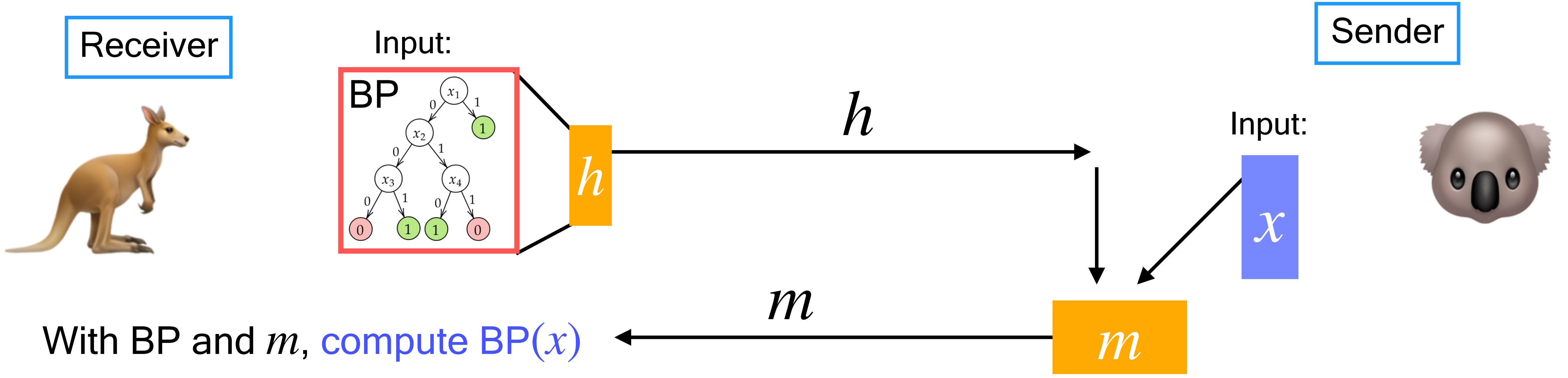


With BP and m, compute BP(x)

$|\text{BP}| \gg |x|$

- Only 1 round of communication allowed (2 messages)
- Communication size grows with:

Laconic Branching Programs



Receiver

Input:

h

m

Sender

Input:

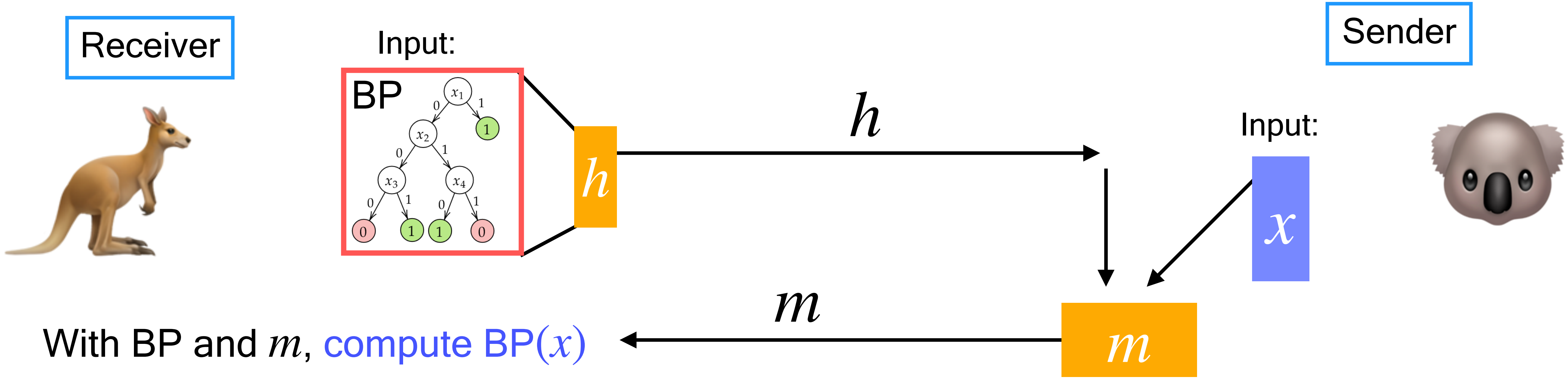
x

With BP and m, compute BP(x)

$|\text{BP}| \gg |x|$

- Only 1 round of communication allowed (2 messages)
- Communication size grows with:
    - the size of the sender's element: $|x|$

Laconic Branching Programs

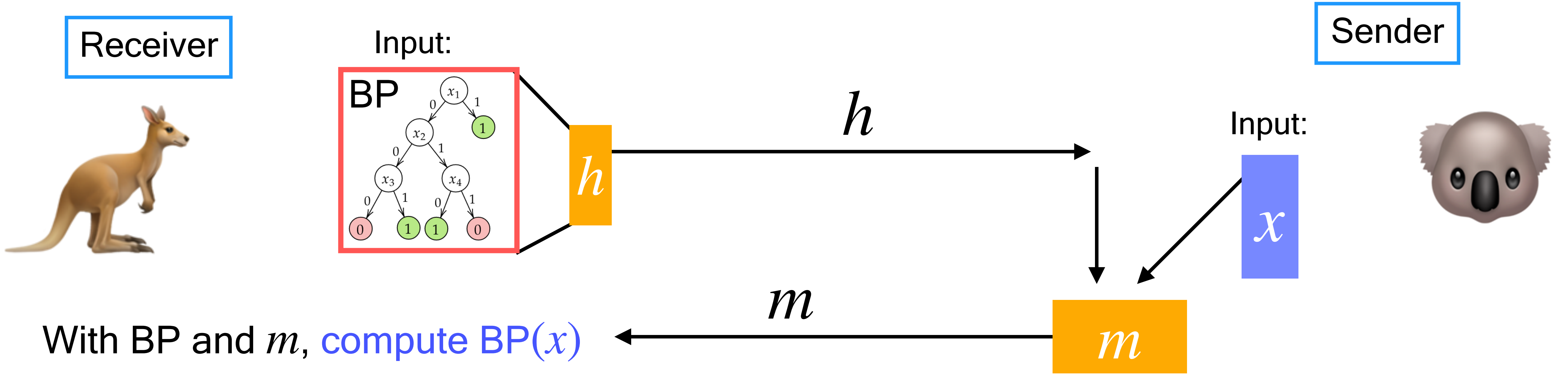


With BP and m, compute BP(x)

$|\text{BP}| \gg |x|$

- Only 1 round of communication allowed (2 messages)
- Communication size grows with:
    - the size of the sender's element: $|x|$
    - the max depth of the receiver's BP

Laconic Branching Programs



With BP and m, compute BP(x)

$|\text{BP}| \gg |x|$

- Only 1 round of communication allowed (2 messages)
- Communication size grows with:
  - the size of the sender's element: $|x|$
  - the max depth of the receiver's BP
- Communication complexity does not otherwise depend on $|\text{BP}|$

Overview

1. Laconic cryptography & previous work
2. Branching programs
3. Laconic branching programs
4. Building blocks:
    1. Garbled circuits
    2. Hash encryption
    3. Garbled circuits + hash encryption
5. Our construction at a high level

Building blocks: Garbled Circuits

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms
- $Garb(C) \rightarrow (\tilde{C}, \{\mathsf{lb}_{i,0}\ ,\ \mathsf{lb}_{i,1}\}_i)$ for all input wires i

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms
- $Garb(C) \to (\tilde{C}, \{\mathsf{lb}_{i,0}\ ,\ \mathsf{lb}_{i,1}\}_i)$ for all input wires i

↳ $$Garb(C) \to \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$$

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms
- $Garb(C) \to (\tilde{C}, \{\mathsf{lb}_{i,0}, \mathsf{lb}_{i,1}\}_i)$ for all input wires i

↳ $$Garb(C) \to \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \ldots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \ldots & \mathsf{lb}_{n,1} \end{bmatrix}\right)$$

2 labels for every input wire:
Label $\mathsf{lb}_{i,0}$ for wire value = 0
Label $\mathsf{lb}_{i,1}$ for wire value = 1

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms
- $Garb(C) \rightarrow (\tilde{C}, \{\mathsf{lb}_{i,0}\ , \ \mathsf{lb}_{i,1}\}_i)$ for all input wires i

$$\hookrightarrow Garb(C) \rightarrow \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$$

2 labels for every input wire:

Label $\mathsf{lb}_{i,0}$ for wire value = 0

Label $\mathsf{lb}_{i,1}$ for wire value = 1

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms
- $Garb(C) \to (\tilde{C}, \{\mathsf{lb}_{i,0}, \mathsf{lb}_{i,1}\}_i)$ for all input wires i

$$\hookrightarrow Garb(C) \to \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$$

- $Eval(\tilde{C}, \{\mathsf{lb}_{i,x_i}\}_i) \to C(x)$

2 labels for every input wire:
Label $\mathsf{lb}_{i,0}$ for wire value = 0
Label $\mathsf{lb}_{i,1}$ for wire value = 1

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms
- $Garb(C) \to (\tilde{C}, \{\mathsf{lb}_{i,0}, \mathsf{lb}_{i,1}\}_i)$ for all input wires i

↳ $$Garb(C) \to \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$$

2 labels for every input wire:
Label $\mathsf{lb}_{i,0}$ for wire value = 0
Label $\mathsf{lb}_{i,1}$ for wire value = 1

- $Eval(\tilde{C}, \{\mathsf{lb}_{i,x_i}\}_i) \to C(x)$

↳ $$Eval(\tilde{C}, [\mathsf{lb}_{1,x_1}, \mathsf{lb}_{2,x_2}, \dots, \mathsf{lb}_{n,x_n}]) \to C(x)$$

One label per input wire. Label 0 or 1 depending on the bits of x

Building blocks: Garbled Circuits

- A garbling scheme consists of garbling, evaluation, and simulation algorithms
- $Garb(C) \rightarrow (\tilde{C}, \{\mathsf{lb}_{i,0}, \mathsf{lb}_{i,1}\}_i)$ for all input wires i

$$\hookrightarrow Garb(C) \rightarrow \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$$

2 labels for every input wire:
Label $\mathsf{lb}_{i,0}$ for wire value = 0
Label $\mathsf{lb}_{i,1}$ for wire value = 1

- $Eval(\tilde{C}, \{\mathsf{lb}_{i,x_i}\}_i) \rightarrow C(x)$

$$\hookrightarrow Eval(\tilde{C}, [\mathsf{lb}_{1,x_1}, \mathsf{lb}_{2,x_2}, \dots, \mathsf{lb}_{n,x_n}]) \rightarrow C(x)$$

One label per input wire. Label 0 or 1 depending on the bits of x

- Security: $(\tilde{C}, \{\mathsf{lb}_{i,x_i}\}_i) \approx_c \mathsf{Sim}(C, C(x))$

Building blocks: Hash Encryption [DG17, BLSV18]

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value

$h = \mathsf{Hash}(x)$ acts like public key

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \mathsf{Hash}(x)$ acts like public key

x acts like secret key

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \mathsf{Hash}(x)$ acts like public key

x acts like secret key

Public parameters are omitted

- Hash function: $\mathsf{Hash}(x) \to h$

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \mathsf{Hash}(x)$ acts like public key

x acts like secret key

Public parameters are omitted

- Hash function: $\mathsf{Hash}(x) \to h$
- Encryption: $c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix}\right)$

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \mathsf{Hash}(x)$ acts like public key

x acts like secret key

Public parameters are omitted

- Hash function: $\mathsf{Hash}(x) \rightarrow h$
- Encryption: $c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix}\right)$
- Decryption: $(m_{1,x_1}, m_{2,x_2}, \dots, m_{n,x_n}) \leftarrow \mathsf{HDec}(x, c)$ if $\mathsf{Hash}(x) = h$

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \text{Hash}(x)$ acts like public key

x acts like secret key

Public parameters are omitted

- Hash function: $\text{Hash}(x) \to h$
- Encryption: $c \leftarrow \text{HEnc}\left(\text{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix}\right)$
- Decryption: $(m_{1,x_1}, m_{2,x_2}, \dots, m_{n,x_n}) \leftarrow \text{HDec}(x, c)$ if $\text{Hash}(x) = h$

Only one $m_{i,b}$ per column is decrypted

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \mathsf{Hash}(x)$ acts like public key

x acts like secret key

Public parameters are omitted

- Hash function: $\mathsf{Hash}(x) \to h$
- Encryption: $c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix}\right)$
- Decryption: $(m_{1,x_1}, m_{2,x_2}, \dots, m_{n,x_n}) \leftarrow \mathsf{HDec}(x, c)$ if $\mathsf{Hash}(x) = h$
- Security: If $x_i = b$, then $m_{i,b-1}$ remains secure, even if $\mathsf{Hash}(x) = h$

Only one $m_{i,b}$ per column is decrypted

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \text{Hash}(x)$ acts like public key

x acts like secret key

Public parameters are omitted

- Hash function: $\text{Hash}(x) \rightarrow h$
- Encryption: $c \leftarrow \text{HEnc}\left(\text{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix}\right)$
- Decryption: $(m_{1,x_1}, m_{2,x_2}, \dots, m_{n,x_n}) \leftarrow \text{HDec}(x, c)$ if $\text{Hash}(x) = h$
- Security: If $x_i = b$, then $m_{i,b-1}$ remains secure, even if $\text{Hash}(x) = h$
  - Note that (i, b) is chosen by the encrypter, not the decrypter

Only one $m_{i,b}$ per column is decrypted

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \mathsf{Hash}(x)$ acts like public key

x acts like secret key

Public parameters are omitted

- Hash function: $\mathsf{Hash}(x) \to h$
- Encryption: $c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \cdots & m_{n,0} \\ m_{1,1} & m_{2,1} & \cdots & m_{n,1} \end{bmatrix}\right)$
- Decryption: $(m_{1,x_1}, m_{2,x_2}, \ldots, m_{n,x_n}) \leftarrow \mathsf{HDec}(x, c)$ if $\mathsf{Hash}(x) = h$
- Security: If $x_i = b$, then $m_{i,b-1}$ remains secure, even if $\mathsf{Hash}(x) = h$
  - Note that (i, b) is chosen by the encrypter, not the decrypter
  - Having the "secret key" x is not sufficient knowledge to decrypt

Only one $m_{i,b}$ per column is decrypted

Building blocks: Hash Encryption [DG17, BLSV18]

- Consists of algorithms Hash, HEnc, HDec
- Encrypt w.r.t. a hash value
- Decrypt with the pre-image of that hash

$h = \mathsf{Hash}(x)$ acts like public key

x acts like secret key

$\}$ Not actually a (pk, sk) pair

Public parameters are omitted

- Hash function: $\mathsf{Hash}(x) \rightarrow h$
- Encryption: $c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix}\right)$
- Decryption: $(m_{1,x_1}, m_{2,x_2}, \dots, m_{n,x_n}) \leftarrow \mathsf{HDec}(x, c)$ if $\mathsf{Hash}(x) = h$
- Security: If $x_i = b$, then $m_{i,b-1}$ remains secure, even if $\mathsf{Hash}(x) = h$
  - Note that (i, b) is chosen by the encrypter, not the decrypter
  - Having the "secret key" x is not sufficient knowledge to decrypt

Only one $m_{i,b}$ per column is decrypted

example

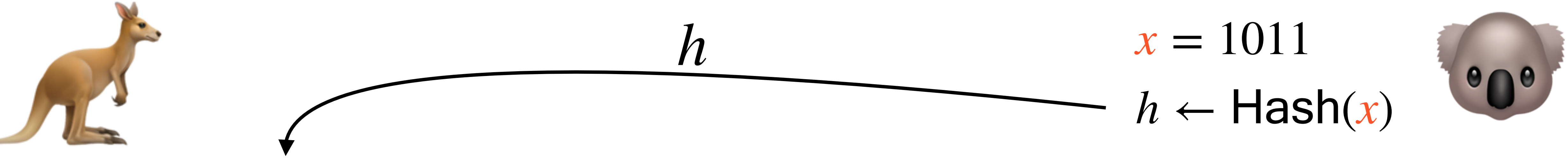
Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

example

Building blocks: Hash Encryption [DG17, BLSV18]



$x = 1011$

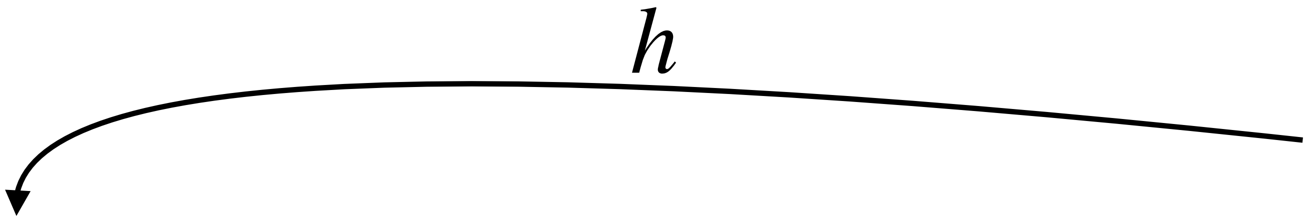
$h \leftarrow \mathsf{Hash}(x)$

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$



$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

$\xleftarrow{h}$

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$\xrightarrow{c}$

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

h

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix} \right)$$

$$\xrightarrow{\quad c \quad}$$

$$(\quad , \quad , \quad , \quad) \leftarrow \mathsf{HDec}(x, c)$$

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

h

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$x = 1 \quad 0 \quad 1 \quad 1$

Decrypt according to the bits of x

c

$$(\quad , \quad , \quad , \quad) \leftarrow \mathsf{HDec}(x, c)$$

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

$\xleftarrow{\quad h \quad}$

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$x = 1 \quad 0 \quad 1 \quad 1$

$\xrightarrow{\quad c \quad}$

$$(m_{1,1}, \quad , \quad , \quad) \leftarrow \mathsf{HDec}(x, c)$$

Decrypt according to the bits of x

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

$$\xleftarrow{\quad h \quad}$$

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$$x = 1 \quad 0 \quad 1 \quad 1$$

Decrypt according to the bits of x

$$\xrightarrow{\quad c \quad} \quad (m_{1,1}, m_{2,0}, \qquad , \qquad) \leftarrow \mathsf{HDec}(x, c)$$

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

$$\xleftarrow{\quad h \quad}$$

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$x = 1 \quad 0 \quad 1 \quad 1$

$$\xrightarrow{\quad c \quad}$$

$$(m_{1,1}, m_{2,0}, m_{3,1}, \qquad) \leftarrow \mathsf{HDec}(x, c)$$

Decrypt according to the bits of x

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

$\xleftarrow{\quad h \quad}$

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$x = 1 \quad 0 \quad 1 \quad 1$

$\xrightarrow{\quad c \quad}$

Decrypt according to the bits of x

$$(m_{1,1}, m_{2,0}, m_{3,1}, m_{4,1}) \leftarrow \mathsf{HDec}(x, c)$$

example

Building blocks: Hash Encryption [DG17, BLSV18]

$\xleftarrow{h}$

$x = 1011$

$h \leftarrow \text{Hash}(x)$

$$c \leftarrow \text{HEnc}\left(\text{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$x = 1 \quad 0 \quad 1 \quad 1$

Decrypt according to the bits of x

$\xrightarrow{c}$

$$(m_{1,1}, m_{2,0}, m_{3,1}, m_{4,1}) \leftarrow \text{HDec}(x, c)$$

- The undecrypted messages $(m_{1,0}, m_{2,1}, m_{3,0}, m_{4,0})$ remain secure

example

Building blocks: Hash Encryption [DG17, BLSV18]

$x = 1011$

$h \leftarrow \mathsf{Hash}(x)$

$\xleftarrow{h}$

$$c \leftarrow \mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & m_{3,0} & m_{4,0} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} \end{bmatrix}\right)$$

$x = 1 \quad 0 \quad 1 \quad 1$

Decrypt according to the bits of x

$\xrightarrow{c}$

$(m_{1,1}, m_{2,0}, m_{3,1}, m_{4,1}) \leftarrow \mathsf{HDec}(x, c)$

- The undecrypted messages $(m_{1,0}, m_{2,1}, m_{3,0}, m_{4,0})$ remain secure
- Can be built from the computational Diffie-Hellman assumption or LWE [DG17, BLSV18]

Garbled Circuits + Hash Encryption

Garbled Circuits + Hash Encryption

- $Garb(C) \rightarrow \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$

Garbled circuit security holds if only one label per column is known

Garbled Circuits + Hash Encryption

- $Garb(C) \rightarrow \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$

Garbled circuit security holds if only one label per column is known

- $\mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix} \right) \rightarrow c$

Hash decryption only reveals one message per column

Garbled Circuits + Hash Encryption

- $Garb(C) \to \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$

Garbled circuit security holds if only one label per column is known

- $\mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix} \right) \to c$

Hash decryption only reveals one message per column

- Hash encrypt the garbled labels:

Garbled Circuits + Hash Encryption

- $Garb(C) \rightarrow \left(\tilde{C}, \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right)$

Garbled circuit security holds if only one label per column is known

- $\mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} m_{1,0} & m_{2,0} & \dots & m_{n,0} \\ m_{1,1} & m_{2,1} & \dots & m_{n,1} \end{bmatrix} \right) \rightarrow c$

Hash decryption only reveals one message per column

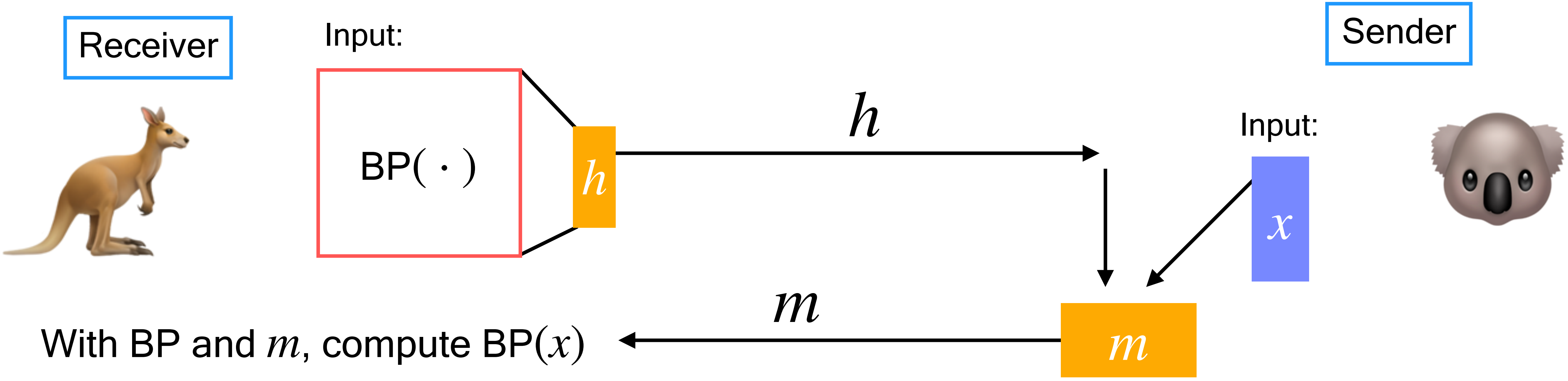
- Hash encrypt the garbled labels:

↳$\mathsf{HEnc}\left(\mathsf{Hash}(x), \begin{bmatrix} \mathsf{lb}_{1,0} & \mathsf{lb}_{2,0} & \dots & \mathsf{lb}_{n,0} \\ \mathsf{lb}_{1,1} & \mathsf{lb}_{2,1} & \dots & \mathsf{lb}_{n,1} \end{bmatrix} \right) \rightarrow c$

Overview

1. Laconic cryptography & previous work
2. Branching programs
3. Laconic branching programs
4. Building blocks:
    1. Garbled circuits
    2. Hash encryption
    3. Garbled circuits + hash encryption
5. Our construction at a high level

Recall: Laconic Branching Programs



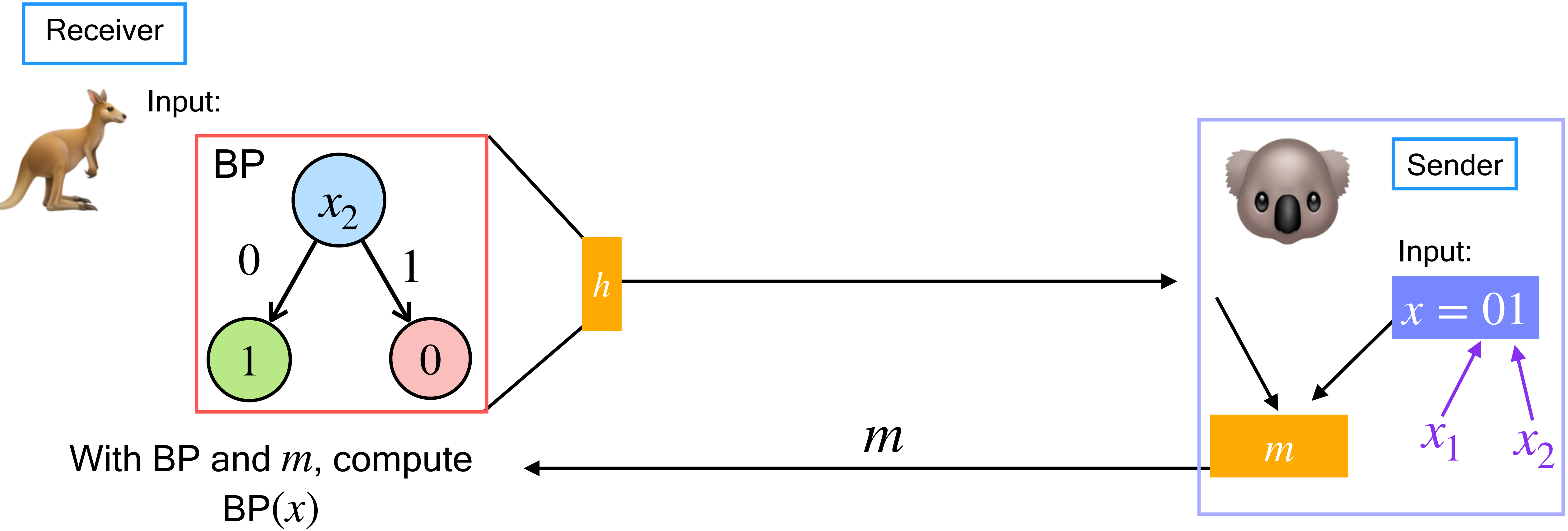
With BP and m, compute BP(x)

$|\text{BP}| \gg |x|$

- Only 1 round of communication allowed (2 messages)
- Communication size grows with:
    - the size of the sender's element: $|x|$
    - the max depth of the receiver's BP
- Communication complexity does not otherwise depend on $|\text{BP}|$

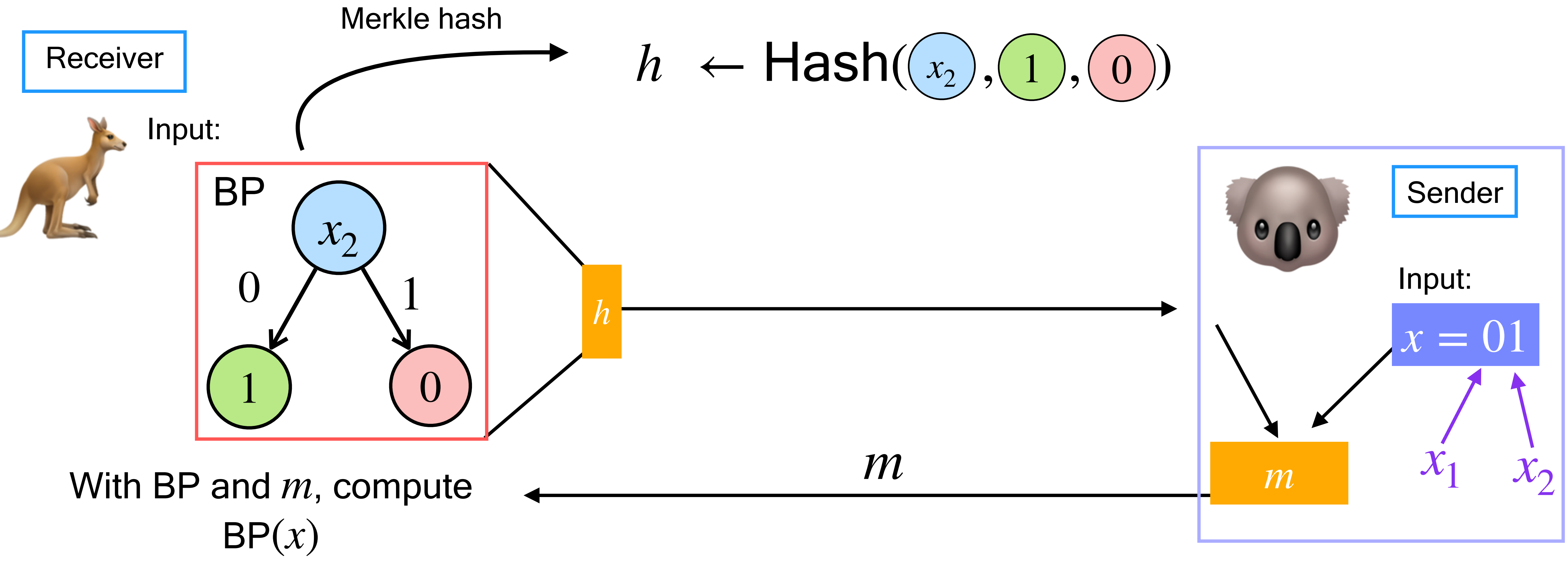
🦘 → 🐨 message

Construction – depth 1 example



With BP and m, compute BP(x)

Construction – depth 1 example



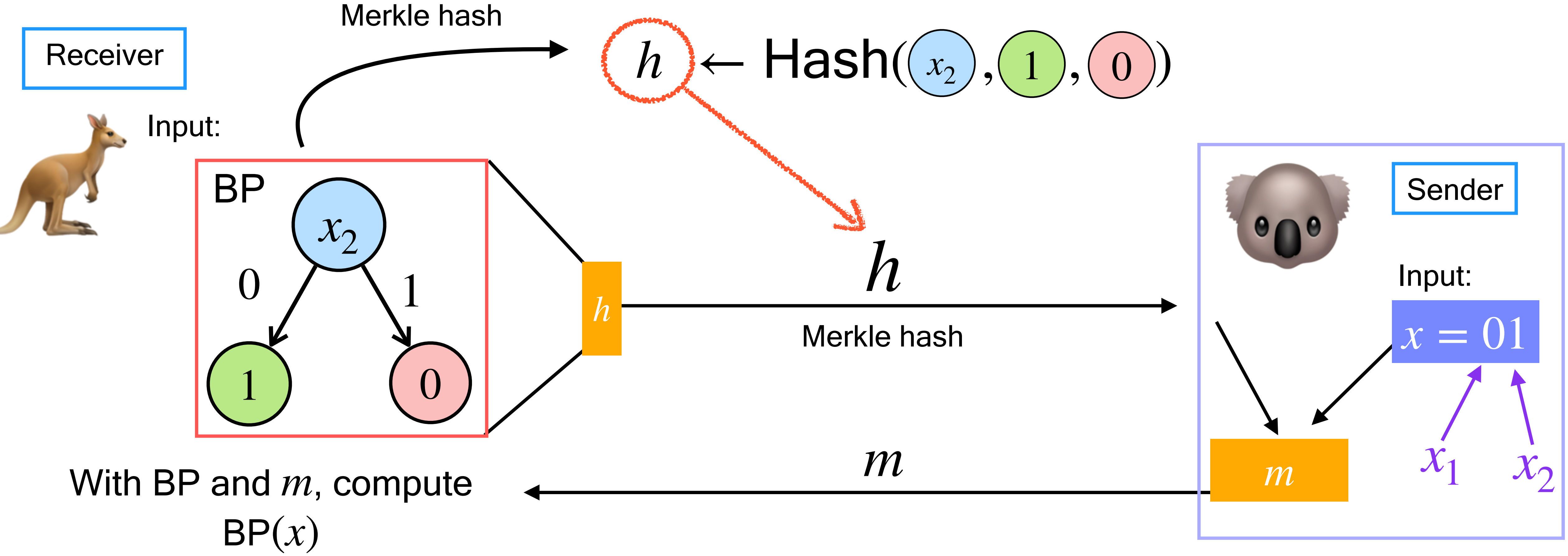
Merkle hash

$$h \leftarrow \text{Hash}(x_2, 1, 0)$$

With BP and m, compute BP(x)

🦘 → 🐨 message

Construction – depth 1 example



Receiver

Input:

Merkle hash

$h \leftarrow$ Hash(x_2, 1, 0)

h

Merkle hash

Sender

Input:

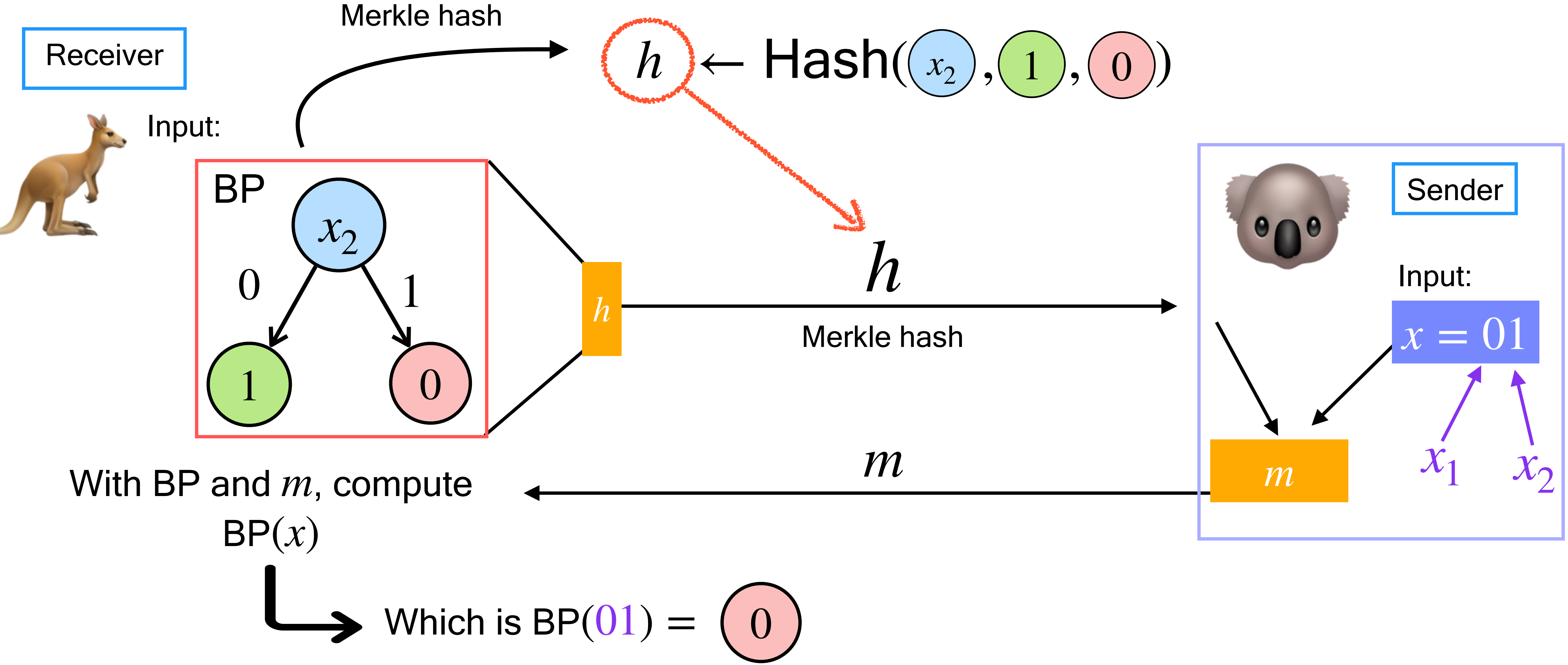
$x = 01$

m

With BP and m, compute BP(x)

🦘 → 🐨 message

Construction – depth 1 example



Merkle hash

Receiver

Input:

BP

x_2

0

1

1

0

$h \leftarrow$ Hash(x_2, 1, 0)

h

h

Merkle hash

Sender

Input:

$x = 01$

x_1 x_2

m

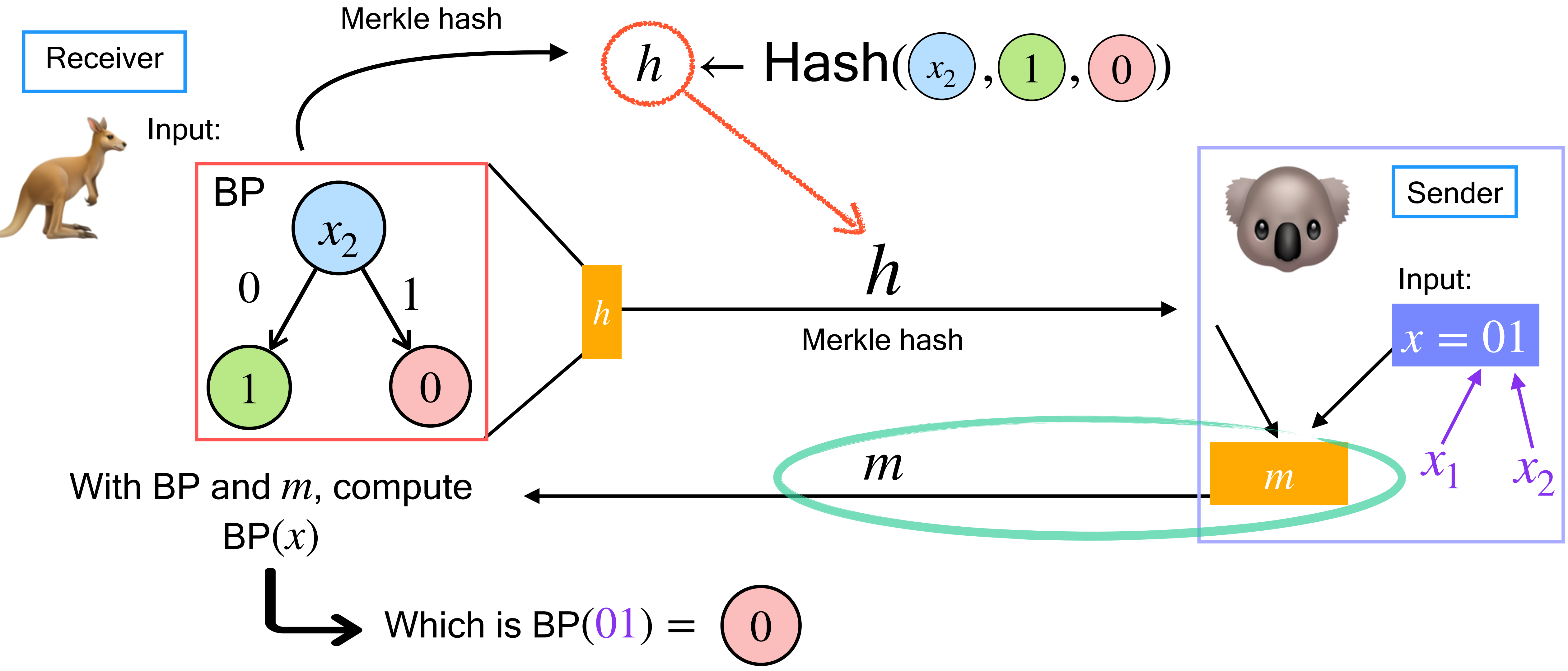
m

With BP and m, compute BP(x)

Which is BP(01) = 0

🦘 → 🐨 message

Construction – depth 1 example



$h \leftarrow$ Hash(x_2, 1, 0)

With BP and m, compute BP(x)

Which is BP(01) = 0

Construction – depth 1 example

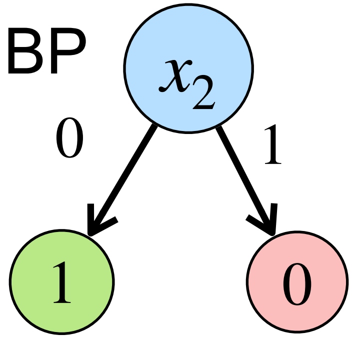
Sender

Input:

$$x = \underset{\uparrow\, x_1}{0}\ \underset{\uparrow\, x_2}{1}$$

Receiver

Input:



🦘 ← 🐨 message

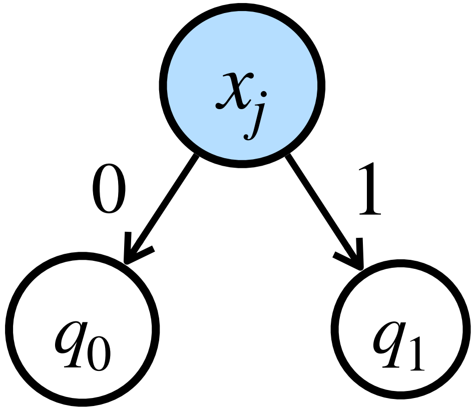
Construction – depth 1 example

Sender

Input:

$$x = \underset{x_1}{\uparrow\!\!0}\ \ \underset{x_2}{\uparrow\!\!1}$$

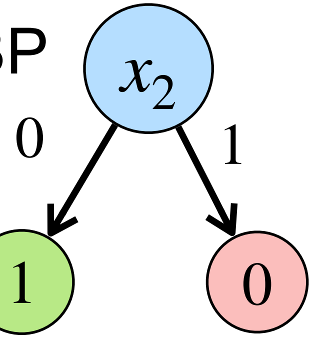
Generic depth 1 BP



Receiver

Input:

BP



🦘 ← 🐨 message

Construction – depth 1 example

Sender

🐨

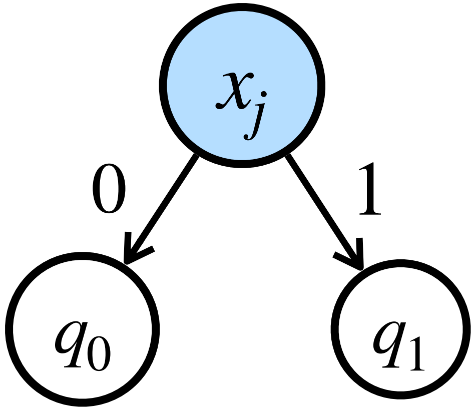
Input:

$$x = \underset{\substack{\uparrow\\ x_1}}{0}\;\;\underset{\substack{\uparrow\\ x_2}}{1}$$

The Sender defines the function:

$$F[x](j, q_0, q_1) \to q_{x_j}$$

Generic depth 1 BP

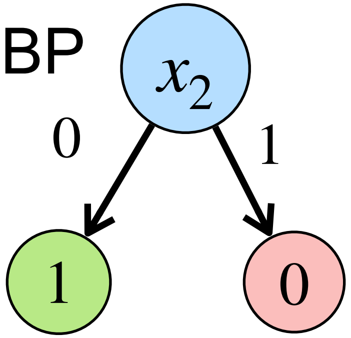


Receiver

Input:

🦘

BP



🦘 ← 🐨 message

Construction – depth 1 example

Sender

🐨

Input:

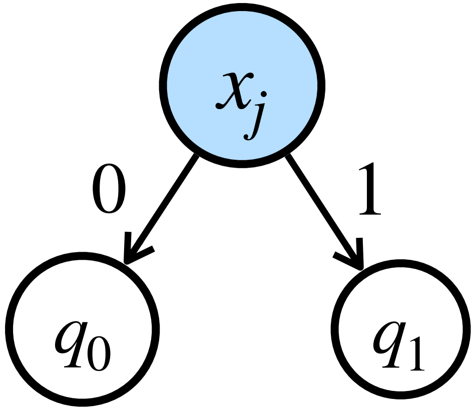
$$x = \underset{\substack{\uparrow\\ x_1}}{0}\ \ \underset{\substack{\uparrow\\ x_2}}{1}$$

The Sender defines the function:

$$F[x](j, q_0, q_1) \to q_{x_j}$$

Outputs q_0 or q_1 depending on the j-th bit of the Sender's input x

Generic depth 1 BP

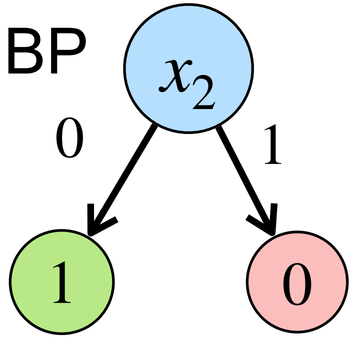


Receiver

Input:

🦘

BP



🦘 ← 🐨 message

Construction – depth 1 example

Sender

Input:

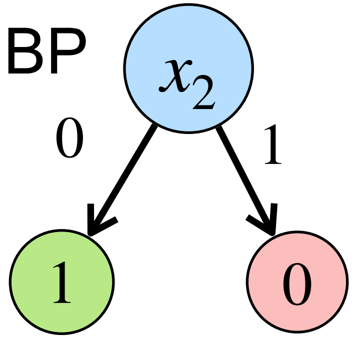
$$x = \underset{\overset{\uparrow}{x_1}}{0}\ \ \underset{\overset{\uparrow}{x_2}}{1}$$

The Sender defines the function:

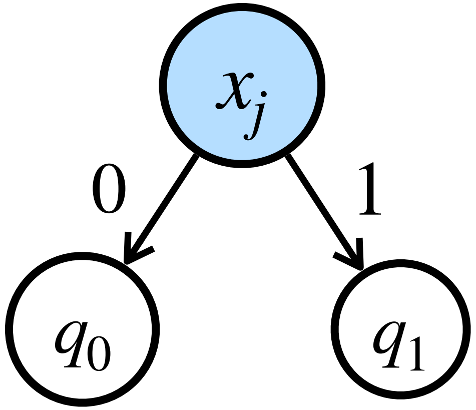
$$F[x](j, q_0, q_1) \rightarrow q_{x_j}$$

Outputs q_0 or q_1 depending on the j-th bit of the Sender's input x

Receiver Input:



Generic depth 1 BP



What if the Receiver could evaluate $F[x]$ on input $(\textcircled{x_2}, \textcircled{1}, \textcircled{0})$?

$$F[x](\textcircled{x_2}, \textcircled{1}, \textcircled{0}) \rightarrow \begin{cases} \textcircled{1} & \text{if } x_2 = 0 \\ \textcircled{0} & \text{if } x_2 = 1 \end{cases}$$

Construction – depth 1 example

Sender

🐨

Input:

$$x = \underset{\underset{x_1}{\uparrow}}{0}\ \underset{\underset{x_2}{\uparrow}}{1}$$

The Sender defines the function:

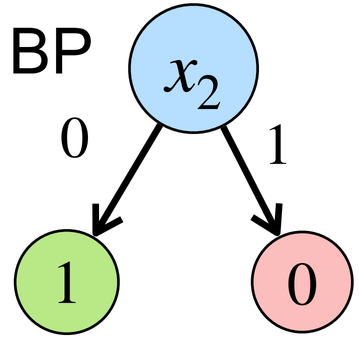
$$F[x](j, q_0, q_1) \rightarrow q_{x_j}$$

Outputs q_0 or q_1 depending on the j-th bit of the Sender's input x

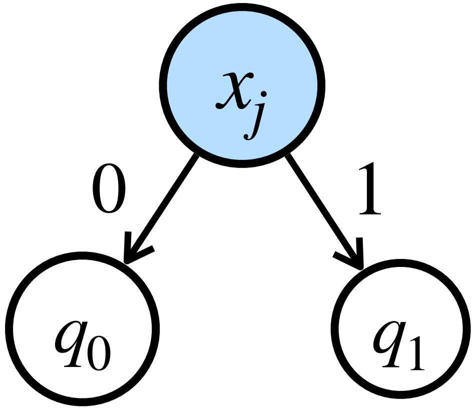
Receiver Input:

🦘

BP



Generic depth 1 BP



What if the Receiver could evaluate $F[x]$ on input $(\textcircled{$x_2$}, \textcircled{1}, \textcircled{0})$?

$$F[x](\textcircled{x_2}, \textcircled{1}, \textcircled{0}) \rightarrow \begin{cases} \textcircled{1} & \text{if } x_2 = 0 \\ \textcircled{0} & \text{if } x_2 = 1 \end{cases}$$

$$\rightarrow \textcircled{0}$$

🦘 ← 🐨 message

Construction – depth 1 example

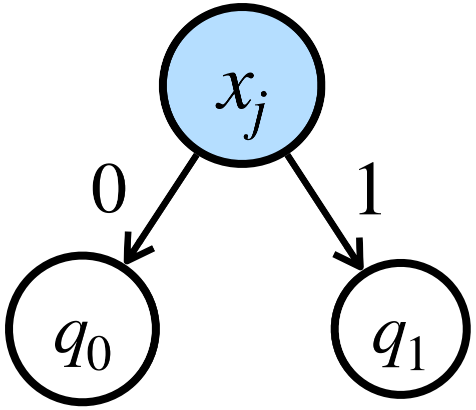
Sender

🐨

Input:

$$x = \underset{x_1}{0}\ \underset{x_2}{1}$$

Generic depth 1 BP



The Sender defines the function:

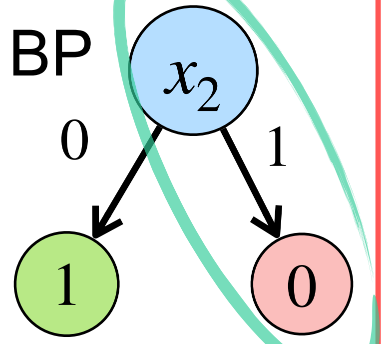
$$F[x](j, q_0, q_1) \rightarrow q_{x_j}$$

Outputs q_0 or q_1 depending on the j-th bit of the Sender's input x

Receiver

🦘

Input:



What if the Receiver could evaluate $F[x]$ on input $(\textcircled{x_2}, \textcircled{1}, \textcircled{0})$?

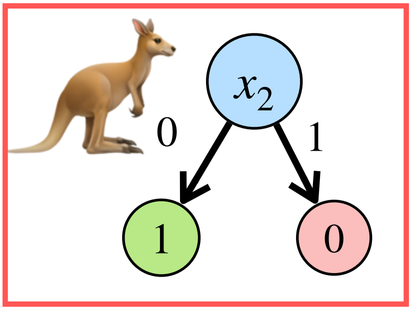
$$F[x](\textcircled{x_2}, \textcircled{1}, \textcircled{0}) \rightarrow \begin{cases} \textcircled{1} & \text{if } x_2 = 0 \\ \textcircled{0} & \text{if } x_2 = 1 \end{cases}$$

$$\rightarrow \textcircled{0}$$

Construction – depth 1 example

Yao’s garbled circuit protocol

Receiver



Sender

x_1 x_2

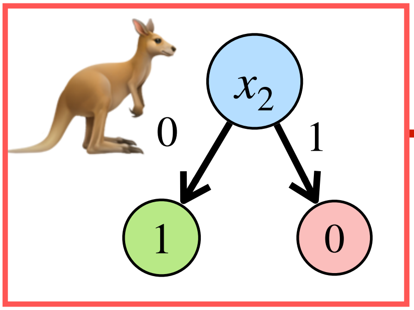
$x = 0\ 1$

$F[x](j, q_0, q_1) \to q_{x_j}$

Construction – depth 1 example

Yao’s garbled circuit protocol

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

Sender

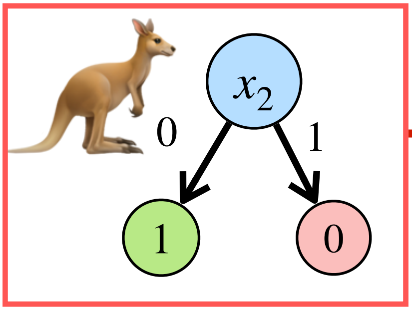
$x = \overset{x_1}{0}\ \overset{x_2}{1}$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

Construction – depth 1 example

Yao’s garbled circuit protocol

Receiver



$$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$$

Sender

$x = \overset{x_1}{0}\ \overset{x_2}{1}$

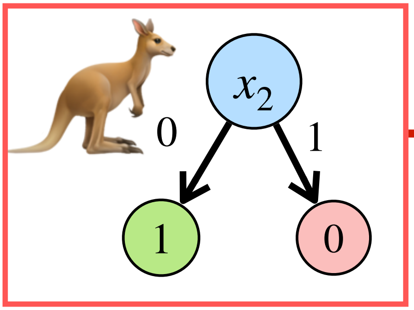
$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

Construction – depth 1 example

Yao’s garbled circuit protocol

Receiver



$$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$$

Sender

x_1 x_2

$$x = 0\ 1$$

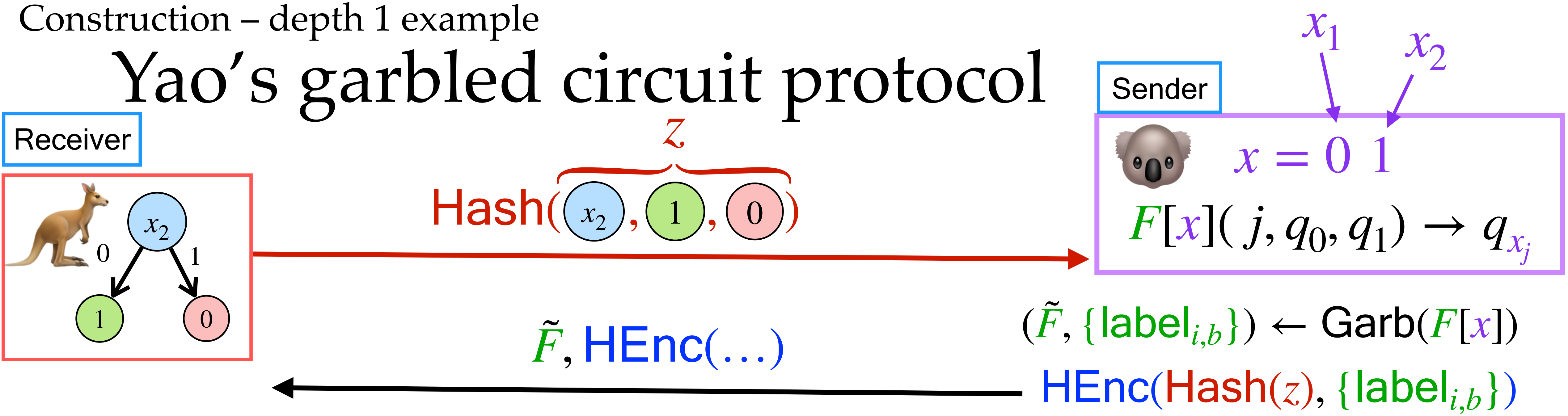
$$F[x](j, q_0, q_1) \rightarrow q_{x_j}$$

$$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$$

$$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$$

Construction – depth 1 example

Yao’s garbled circuit protocol



$x = 0\ 1$

$F[x](j, q_0, q_1) \to q_{x_j}$

$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

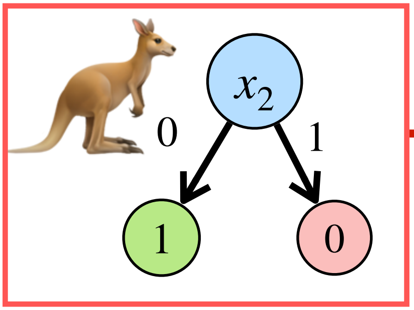
$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\tilde{F}, \text{HEnc}(\ldots)$

Construction – depth 1 example

Yao's garbled circuit protocol

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

Sender

🐨 $x = \overset{x_1}{0}\ \overset{x_2}{1}$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\tilde{F}$, HEnc(…)

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$$\parallel$$

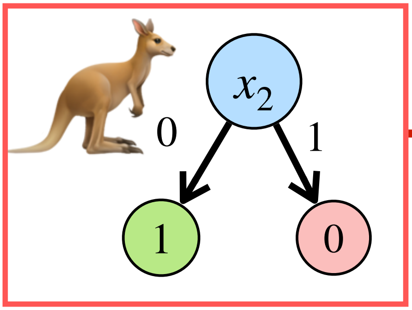
$$\text{HDec}((\overbrace{x_2, 1, 0}^{z}), \text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\}))$$

$$\Downarrow$$

Construction – depth 1 example

Yao’s garbled circuit protocol

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

Sender

$x = \overset{x_1}{0}\ \overset{x_2}{1}$

$F[x](j, q_0, q_1) \to q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

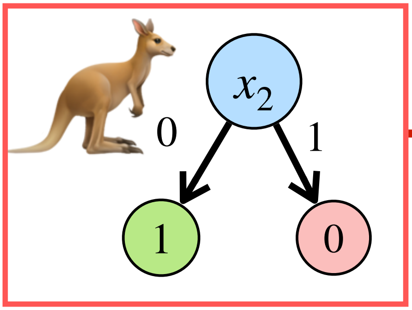
$\tilde{F}, \text{HEnc}(\ldots)$

$$\begin{gathered} \shortparallel \\ \text{HDec}((\overbrace{x_2, 1, 0}^{z}), \text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})) \\ \Downarrow \\ \{\text{label}_{i,z[i]}\} \end{gathered}$$

Construction – depth 1 example

Yao's garbled circuit protocol

Receiver



Sender

x_1 x_2

$x = 0\ 1$

$F[x](j, q_0, q_1) \to q_{x_j}$

$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\tilde{F}, \text{HEnc}(\ldots)$

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\|$

$\text{HDec}((\overbrace{x_2, 1, 0}^{z}), \text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\}))$

$\Downarrow$

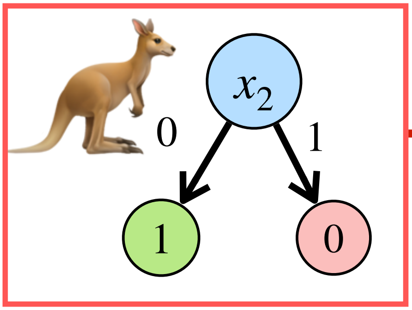
$\{\text{label}_{i,z[i]}\}$

Then evaluate garbled circuit:

Construction – depth 1 example

Yao's garbled circuit protocol

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

Sender

$x = \overset{x_1}{0}\ \overset{x_2}{1}$

$F[x](j, q_0, q_1) \to q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\tilde{F}$, HEnc(…)

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\Vert$

$\text{HDec}(\overbrace{(x_2, 1, 0)}^{z}, \text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\}))$

$\Downarrow$

$\{\text{label}_{i,z[i]}\}$

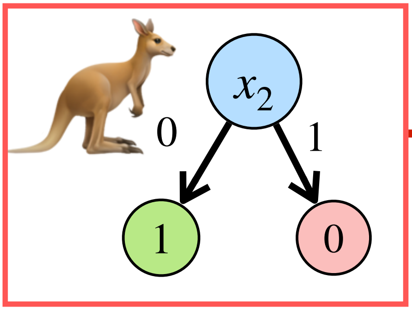
Then evaluate garbled circuit:

$\text{Eval}(\tilde{F}, \{\text{label}_{i,z[i]}\})$

Construction – depth 1 example

Yao’s garbled circuit protocol

Receiver



$$\text{Hash}\overbrace{(\textcircled{x_2}, \textcircled{1}, \textcircled{0})}^{z}$$

Sender

$$x = \overset{x_1}{0}\ \overset{x_2}{1}$$

$$F[x](j, q_0, q_1) \rightarrow q_{x_j}$$

$$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$$

$$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$$

$\tilde{F}$, HEnc(…)

$$\parallel$$

$$\text{HDec}(\overbrace{(\textcircled{x_2}, \textcircled{1}, \textcircled{0})}^{z}, \text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\}))$$

$$\Downarrow$$

$$\{\text{label}_{i,z[i]}\}$$

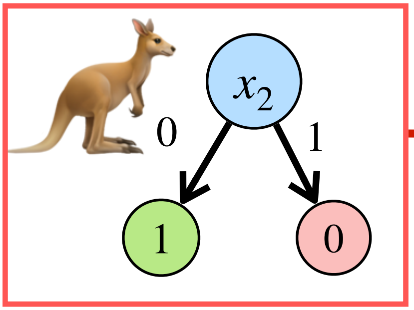
Then evaluate garbled circuit:

$$\text{Eval}(\tilde{F}, \{\text{label}_{i,z[i]}\}) \longrightarrow \textcircled{0}$$

Construction – depth 1 example

Yao’s garbled circuit protocol

Receiver



$\text{Hash}(\overbrace{\textcircled{x_2}, \textcircled{1}, \textcircled{0}}^{z})$

Sender

$x = \overset{x_1}{0}\ \overset{x_2}{1}$

$F[x](j, q_0, q_1) \to q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\tilde{F}$, HEnc(…)

$$\| \\ \text{HDec}((\overbrace{\textcircled{x_2}, \textcircled{1}, \textcircled{0}}^{z}), \text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\}))$$

$$\Downarrow$$

$\{\text{label}_{i,z[i]}\}$

$$F[x](\overbrace{\textcircled{x_2}, \textcircled{1}, \textcircled{0}}^{z}) \to \begin{cases} \textcircled{1} & \text{if } x_2 = 0 \\ \textcircled{0} & \text{if } x_2 = 1 \end{cases}$$

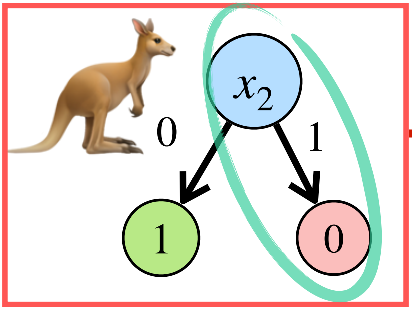
Then evaluate garbled circuit:

$\text{Eval}(\tilde{F}, \{\text{label}_{i,z[i]}\}) \longrightarrow \textcircled{0}$

Construction – depth 1 example

Yao's garbled circuit protocol

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

Sender

$x = \overset{x_1}{0}\,\overset{x_2}{1}$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

$\tilde{F}$, HEnc(…)

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$$\text{HDec}((\overbrace{x_2, 1, 0}^{z}), \text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\}))$$

$$\Downarrow$$

$$\{\text{label}_{i,z[i]}\}$$

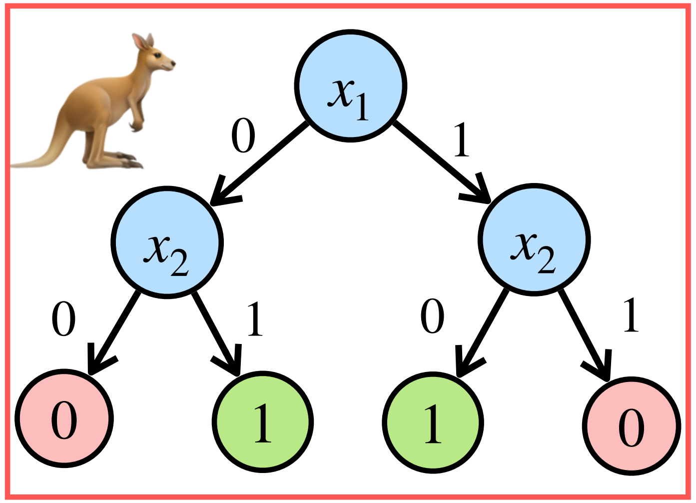
Then evaluate garbled circuit:

$$\text{Eval}(\tilde{F}, \{\text{label}_{i,z[i]}\}) \longrightarrow 0$$

$$F[x](\overbrace{x_2, 1, 0}^{z}) \rightarrow \begin{cases} 1 & \text{if } x_2 = 0 \\ 0 & \text{if } x_2 = 1 \end{cases}$$

Yao's garbled circuit protocol – Depth 2?

Receiver



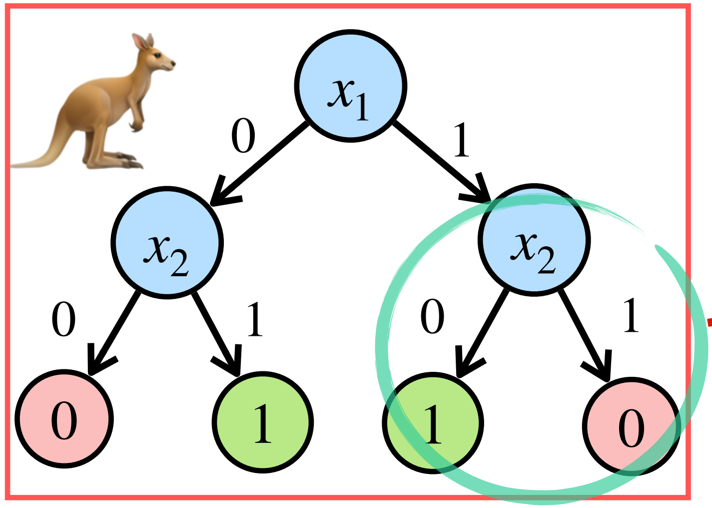
Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

Yao’s garbled circuit protocol – Depth 2?

Receiver



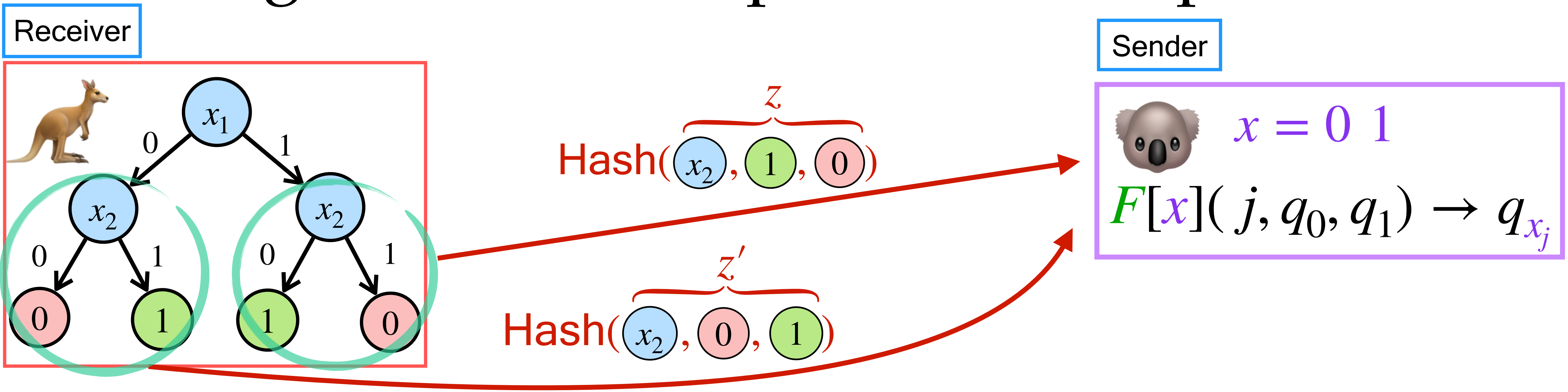
Hash$(\overbrace{x_2, 1, 0}^{z})$

Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

Yao’s garbled circuit protocol – Depth 2?



Receiver

$$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$$

$$\text{Hash}(\overbrace{x_2, 0, 1}^{z'})$$

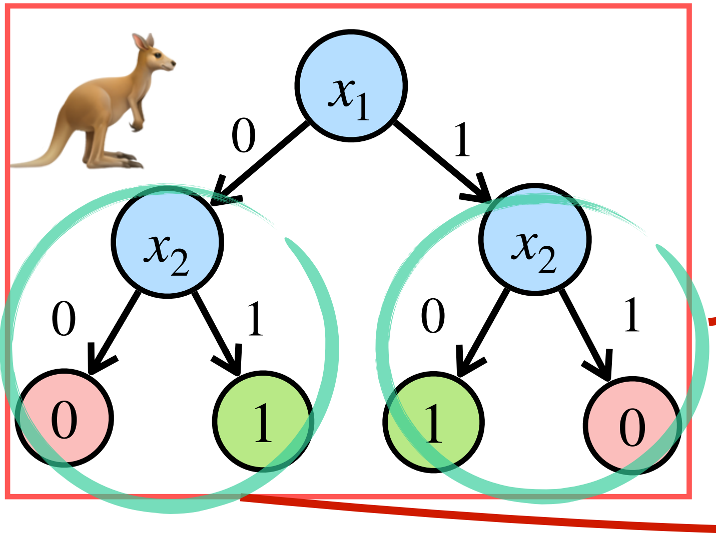
Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

Yao's garbled circuit protocol – Depth 2?

Receiver



Hash($\overbrace{x_2, 1, 0}^{z}$)

Hash($\overbrace{x_2, 0, 1}^{z'}$)

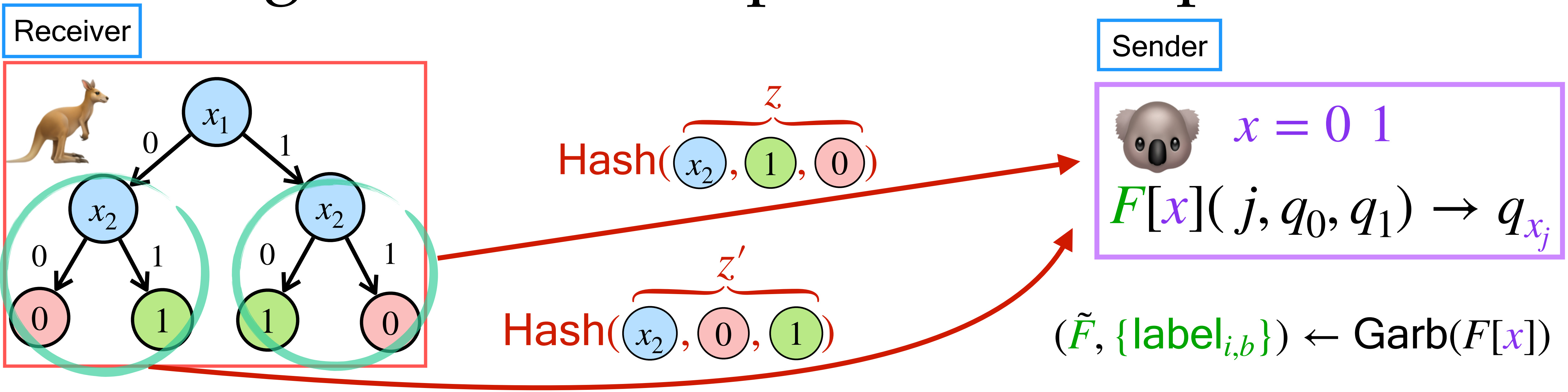
Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \to q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

Yao's garbled circuit protocol – Depth 2?



Hash($\overbrace{x_2, 1, 0}^{z}$)

Hash($\overbrace{x_2, 0, 1}^{z'}$)

Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

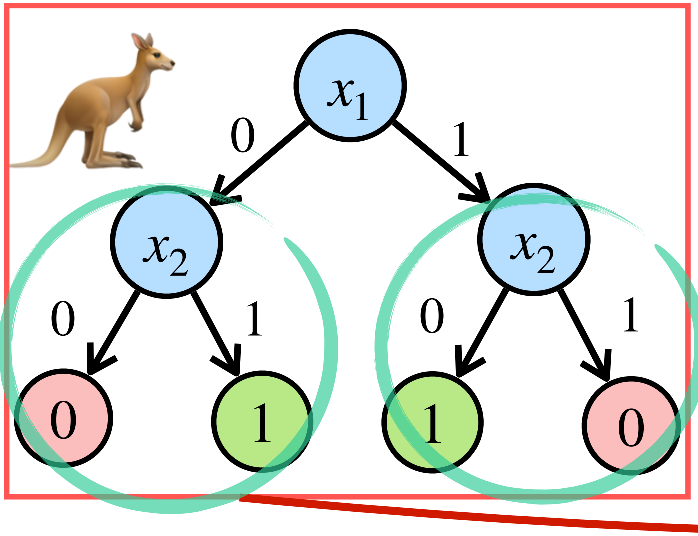
$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\text{HEnc}(\text{Hash}(z'), \{\text{label}_{i,b}\})$

Yao’s garbled circuit protocol – Depth 2?

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

$\text{Hash}(\overbrace{x_2, 0, 1}^{z'})$

Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

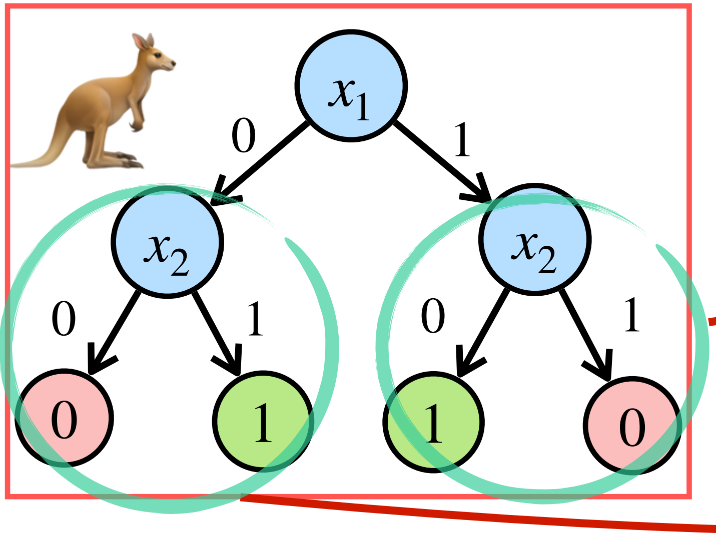
$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\text{HEnc}(\text{Hash}(z'), \{\text{label}_{i,b}\})$

$\tilde{F}, \text{HEnc}(\ldots z \ldots), \text{HEnc}(\ldots z' \ldots)$

Yao’s garbled circuit protocol – Depth 2?

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

$\text{Hash}(\overbrace{x_2, 0, 1}^{z'})$

Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \to q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

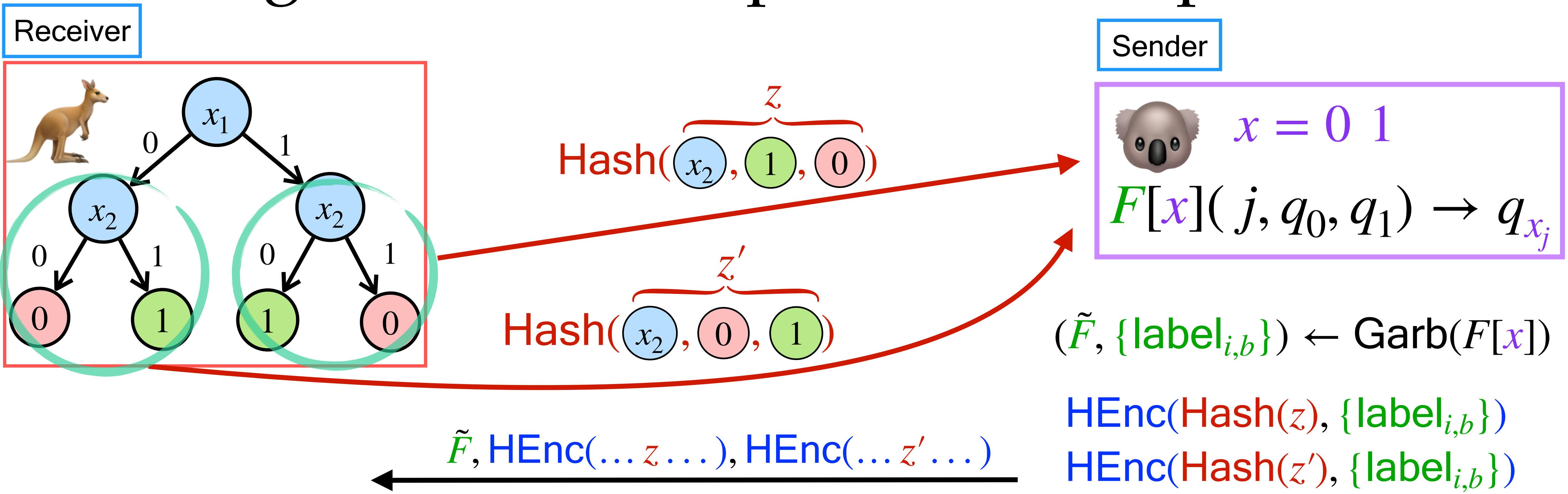
$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\text{HEnc}(\text{Hash}(z'), \{\text{label}_{i,b}\})$

$\tilde{F}, \text{HEnc}(\ldots z \ldots), \text{HEnc}(\ldots z' \ldots)$

But sending encryptions wrt. both z and z' would destroy garbled circuit security.

Yao’s garbled circuit protocol – Depth 2?



Receiver

Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \to q_{x_j}$

Hash$(\overbrace{x_2, 1, 0}^{z})$

Hash$(\overbrace{x_2, 0, 1}^{z'})$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

HEnc(Hash(z), $\{\text{label}_{i,b}\}$)

HEnc(Hash(z'), $\{\text{label}_{i,b}\}$)

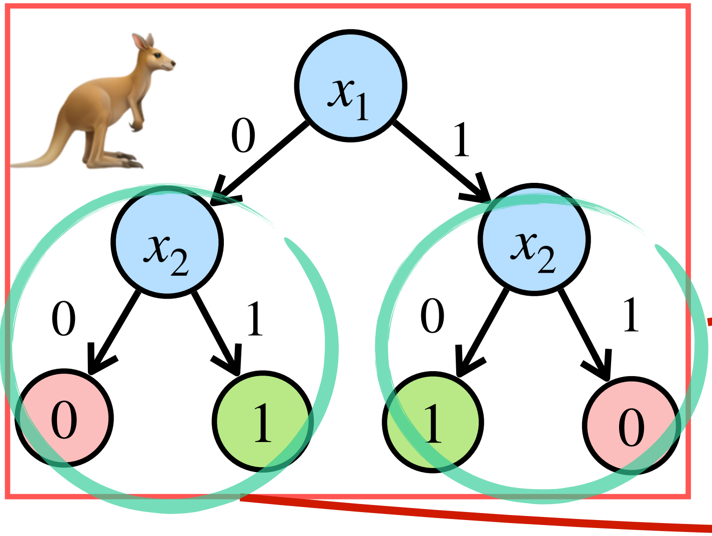
$\tilde{F}$, HEnc($\ldots z \ldots$), HEnc($\ldots z' \ldots$)

But sending encryptions wrt. both z and z' would destroy garbled circuit security.

And sending both Hash(z) and Hash(z') would cause communication cost blow-up (grow with BP size, not BP depth)

Yao's garbled circuit protocol – Depth 2?

Receiver



$\text{Hash}(\overbrace{x_2, 1, 0}^{z})$

$\text{Hash}(\overbrace{x_2, 0, 1}^{z'})$

Sender

$x = 0\ 1$

$F[x](j, q_0, q_1) \rightarrow q_{x_j}$

$(\tilde{F}, \{\text{label}_{i,b}\}) \leftarrow \text{Garb}(F[x])$

$\text{HEnc}(\text{Hash}(z), \{\text{label}_{i,b}\})$

$\text{HEnc}(\text{Hash}(z'), \{\text{label}_{i,b}\})$

$\tilde{F}, \text{HEnc}(\ldots z \ldots), \text{HEnc}(\ldots z' \ldots)$

But sending encryptions wrt. both z and z' would destroy garbled circuit security.

And sending both Hash(z) and Hash(z') would cause communication cost blow-up (grow with BP size, not BP depth)

We use deferred encryption to fixed these problems – see the paper for details!

Summary

Summary

- New construction for laconic 2PC of branching programs from LWE or CDH

Summary

- New construction for laconic 2PC of branching programs from LWE or CDH
- First laconic BP construction from an assumption other than LWE

Summary

- New construction for laconic 2PC of branching programs from LWE or CDH
- First laconic BP construction from an assumption other than LWE

[ABD+21]
Laconic PSI
from LWE or CDH

[This work]
Laconic BPs
from LWE or CDH

[QWW18]
General laconic 2PC
from LWE

Less general

More general

Summary

- New construction for laconic 2PC of branching programs from LWE or CDH
- First laconic BP construction from an assumption other than LWE

[ABD+21]
Laconic PSI
from LWE or CDH

[This work]
Laconic BPs
from LWE or CDH

[QWW18]
General laconic 2PC
from LWE

Less general

More general

- Can realise private set intersection and private set union

Summary

- New construction for laconic 2PC of branching programs from LWE or CDH
- First laconic BP construction from an assumption other than LWE

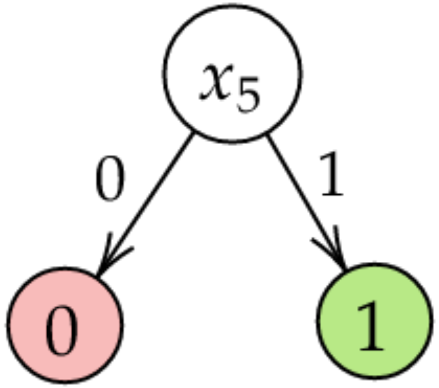
[ABD+21]
Laconic PSI
from LWE or CDH

[This work]
Laconic BPs
from LWE or CDH

[QWW18]
General laconic 2PC
from LWE

Less general → More general

- Can realise private set intersection and private set union
- Wildcards allow receiver's set to be represented concisely



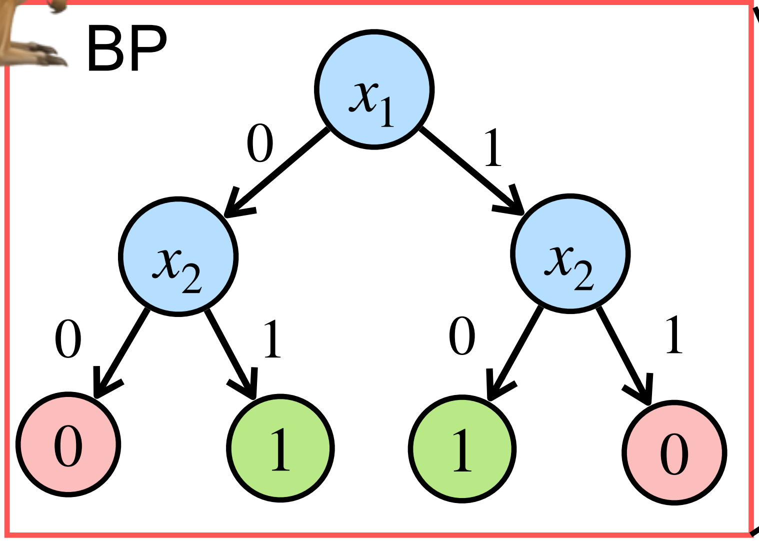
Backup

🦘 → 🐨 message

Construction – depth 2 example

Receiver

Input:



Merkle hash

$$h_{\text{root}} = \text{Hash}(x_1, h_0, h_1)$$

$$h_0 = \text{Hash}(x_2, 0, 1) \qquad h_1 = \text{Hash}(x_2, 1, 0)$$

h

h_{root}

Merkle hash

Sender

Input:

$x = 01$

m

m

With BP and m, compute BP(x)

Which is BP$(01) = 1$

Backup

Construction – depth 2 example

🦘 ← 🐨 message

Sender

🐨

Input: $x = 01$

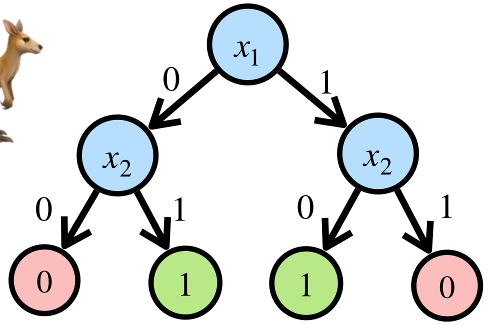
The Sender defines the function:

$$F[x](j, q_0, q_1) \to q_{x_j}$$

Outputs q_0 or q_1 depending on the j-th bit of the Sender's input x

Receiver Input: BP

🦘



The Sender defines another function:

$$V[x, \{\text{lb}_{i,b}\}](u, h'_0, h'_1) = \begin{cases} \text{HEnc}(h'_0, \{\text{lb}_{i,b}\}) & \text{if } x_u = 0 \\ \text{HEnc}(h'_1, \{\text{lb}_{i,b}\}) & \text{if } x_u = 1 \end{cases}$$

Outputs a Hash encryption of garbled circuit labels $\{\text{lb}_{i,b}\}$ wrt. h'_{x_u}

deferred encryption: labels $\{\text{lb}_{i,b}\}$ for $\tilde{F}$ are hash-encrypted when $V[x, \{\text{lb}_{i,b}\}]$ is evaluated.

Backup

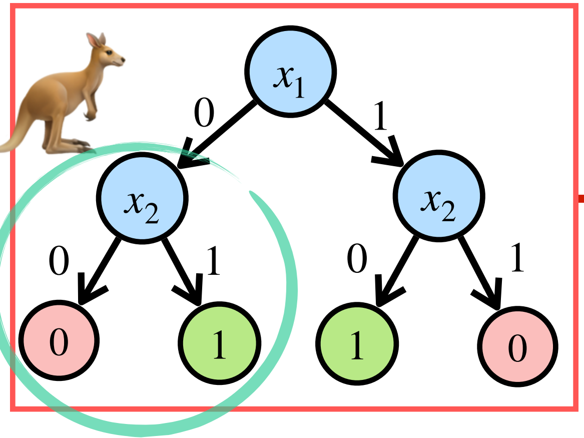
Yao's garbled circuit protocol – depth 2

Sender

🐨 $x = \boxed{0}1$ $F[x](j, q_0, q_1) \rightarrow q_{x_j}$

$$V[x, \{\mathsf{lb}_{i,b}\}](u, h'_0, h'_1) = \begin{cases} \mathsf{HEnc}(h'_0, \{\mathsf{lb}_{i,b}\}) & \text{if } x_u = 0 \\ \mathsf{HEnc}(h'_1, \{\mathsf{lb}_{i,b}\}) & \text{if } x_u = 1 \end{cases}$$

Receiver



$$h_{\text{root}} = \mathsf{Hash}(\overbrace{x_1, h_0, h_1}^{z_{\text{root}}}) \longrightarrow$$

$$\longleftarrow \tilde{F}, \tilde{V}, \mathsf{HEnc}(\ldots)$$

$(\tilde{F}, \{\mathsf{lb}_{i,b}\}) \leftarrow \mathsf{Garb}(F[x])$

$(\tilde{V}, \{\mathsf{lb}_{i,b}\}) \leftarrow \mathsf{Garb}(V[x, \{\mathsf{lb}_{i,b}\}])$

$\mathsf{HEnc}(\mathsf{Hash}(z_{\text{root}}), \{\mathsf{lb}_{i,b}\})$

$$\|$$

$$\mathsf{HDec}((\overbrace{x_1, h_0, h_1}^{z_{\text{root}}}), \mathsf{HEnc}(\mathsf{Hash}(z_{\text{root}}), \{\mathsf{lb}_{i,b}\}))$$

$$\Downarrow$$

$$\{\mathsf{lb}_{i,z_{\text{root}}[i]}\}$$

Then evaluate the V garbled circuit:

$h_0 = \mathsf{Hash}(x_2, 0, 1)$

$$\mathsf{Eval}(\tilde{V}, \{\mathsf{lb}_{i,z_{\text{root}}[i]}\}) \implies V[x, \{\mathsf{lb}_{i,b}\}](x_1, h_0, h_1) = \begin{cases} \mathsf{HEnc}(h_0, \{\mathsf{lb}_{i,b}\}) & \text{if } x_1 = 0 \\ \mathsf{HEnc}(h_1, \{\mathsf{lb}_{i,b}\}) & \text{if } x_1 = 1 \end{cases}$$

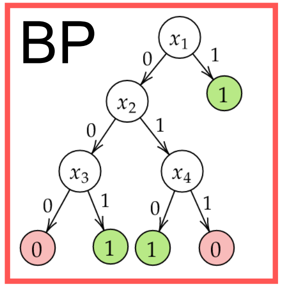
This brings the Receiver to the depth 1 case. Using $\mathsf{HEnc}(h_0, \{\mathsf{lb}_{i,b}\})$ and $\tilde{F}$, 🦘 can finish the computation of BP(01).

Can we use Fully Homomorphic Encryption?

attempt 1

Receiver

Input:



Sender

Input: x

$(pk, sk) \leftarrow \text{FHE.Gen}(1^\lambda)$

$c \leftarrow \text{FHE.Enc}(pk, x)$

$\xleftarrow{(c, pk)}$

$c' \leftarrow \text{FHE.Eval}(pk, \text{BP}, c)$

$\xrightarrow{c'}$

$\text{BP}(x) \leftarrow \text{FHE.Dec}(sk, c')$

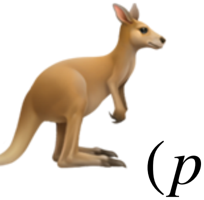
But we want the party with larger input (🦘) to learn the output BP(x) first, not the party with smaller input (🐨)

Backup

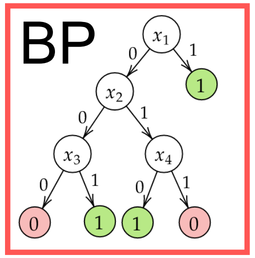
attempt 2

Can we use Fully Homomorphic Encryption?

Receiver



Input:



Sender

Input: x

$(pk, sk) \leftarrow \text{FHE.Gen}(1^\lambda)$

$c \leftarrow \text{FHE.Enc}(pk, \text{BP})$

$$(pk, c) \longrightarrow$$

$c' \leftarrow \text{FHE.Eval}(pk, c, x)$

Encryption of BP$(\cdot)$

$$\xleftarrow{\quad c' \quad}$$

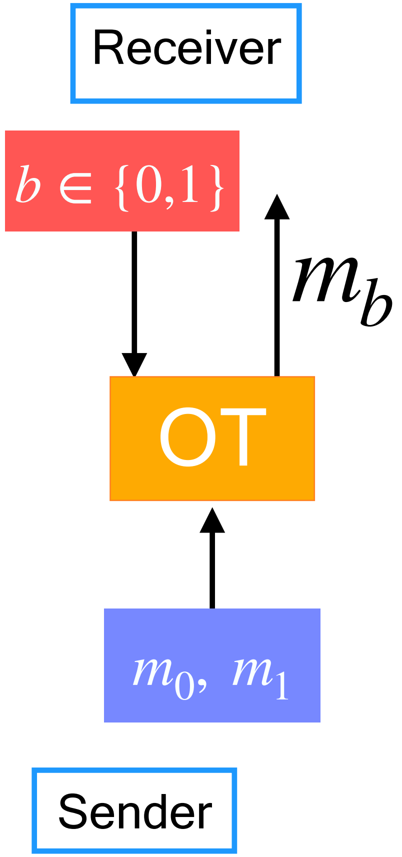
$\text{BP}(x) \leftarrow \text{FHE.Dec}(sk, c')$

But the size of the ciphertext,
$c \leftarrow \text{FHE.Enc}(pk, \text{BP})$, depends on $|\text{BP}|$,
violating laconicism requirements

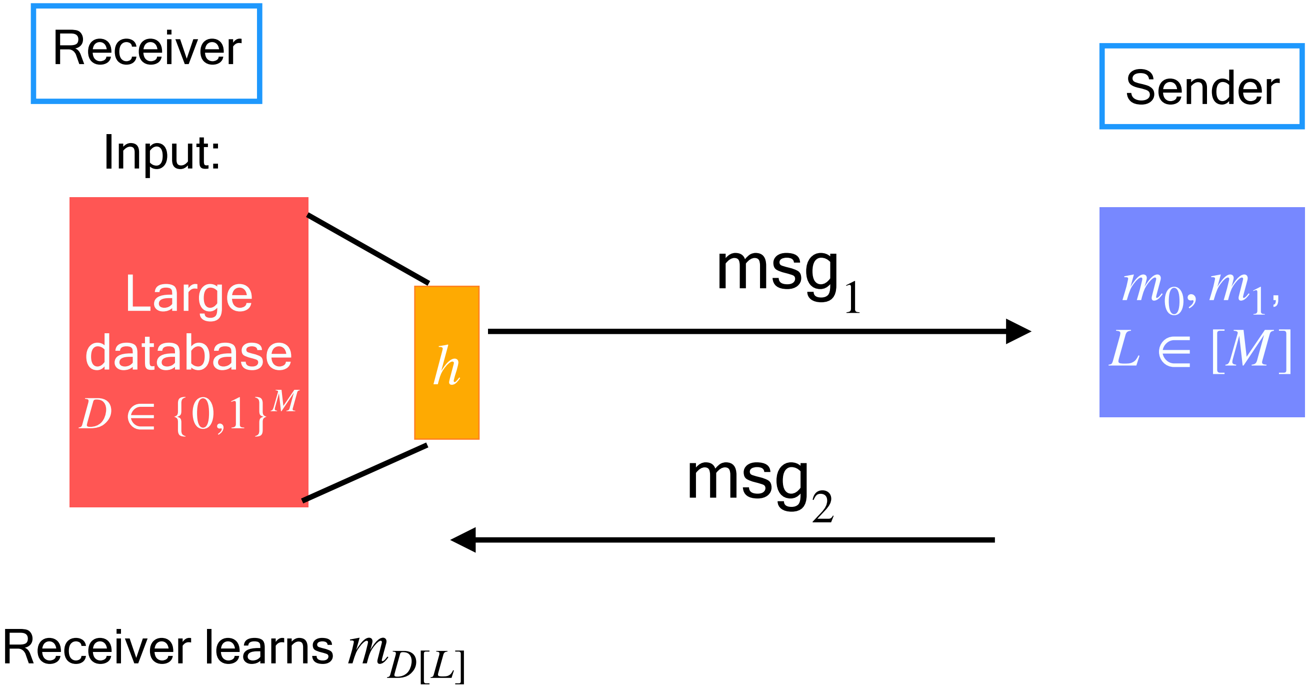
Backup

Laconic Oblivious Transfer (OT)

- Regular oblivious transfer:



- Laconic oblivious transfer:



Receiver learns $m_{D[L]}$

Backup: Construction of anonymous hash encryption from CDH [BLSV18]

- Algorithms: Setup, Gen, SingleEnc, SingleDec
- Setup($1^\lambda, 1^n$): Let $(\mathbb{G}, g, q) \leftarrow \mathcal{G}(1^\lambda)$ and $\alpha_{i,b} \leftarrow \mathbb{Z}_q$ for $i \in [n]$ and $b \in \{0,1\}$. Output crs $= ((\mathbb{G}, g, q), \{g^{\alpha_{i,b}}\}_{i,b})$.
- Gen(crs, x): Output $h = \prod_{i}^{n} g^{\alpha_{i,x_i}}$
- SingleEnc(crs, $h, i, \boldsymbol{m}$): Let $r \leftarrow \mathbb{Z}_q$, $\hat{g}^{\alpha_{i,b}} = h^r g^{-r\,\alpha_{i,b}}$, and $\mu_{i,b} =$ gl-enc($\hat{g}^{\alpha_{i,b}}, \boldsymbol{m}_i$). $\forall\ b \in \{0,1\}, j \neq i$, let $\hat{g}^{\alpha_{j,b}} = g^{r\,\alpha_{j,b}}$. Output ct $= (\{\hat{g}^{j,b}\}_{j \neq i, b}, \{\mu_{i,b}\}_b)$.
- SingleDec(crs, x, i, ct): Let $\hat{g}^{\alpha_{i,x_i}} = \prod_{j \neq i} \hat{g}^{\alpha_{j,x_j}}$. Output gl-dec($\hat{g}^{\alpha_{i,x_i}}, \mu_{i,x_i}$).

gl-enc$(x, b) := (\alpha, \langle \alpha, x \rangle \oplus b),\ \alpha \xleftarrow{\$} \{0,1\}^n$

gl-dec$(x, (\alpha, \sigma)) := \sigma \oplus \langle \alpha, x \rangle$