Garbled Circuits

CS 598 DH

Today’s objectives

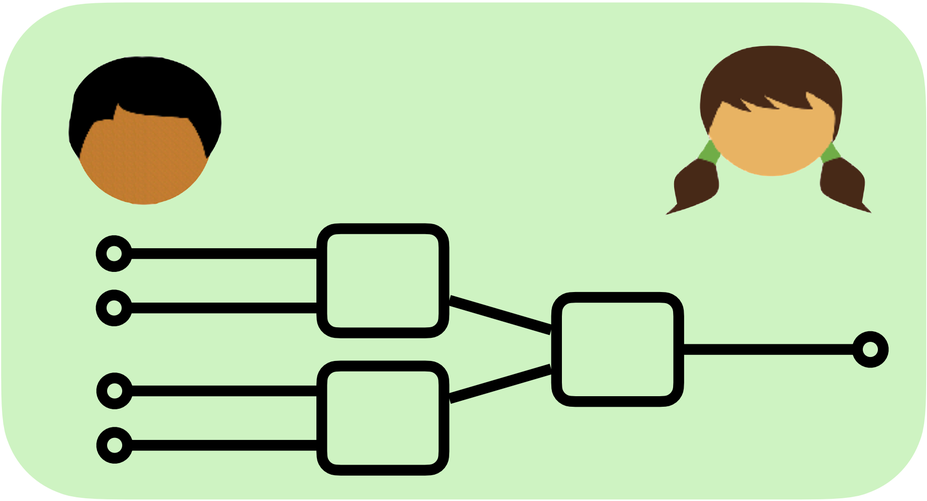
Review GMW and its round complexity

Introduce Garbled Circuits

Discuss trade-offs between GMW and GC

Explore GC security proof

GMW Protocol



Propagate secret shares from input wires to output wires

Use OT to implement AND gates

Cost:

$O(|C|)$ OTs

Number of protocol rounds scales with multiplicative depth of C

View$_{\text{Alice}}^{\text{GMW(C)}}$(x,y):

for g in C; **switch** on g:

case INPUT[Alice](w):

$w^A \leftarrow\$ \{0,1\}$

$w^B \leftarrow$ “next bit of x” $\oplus$ w^A

case INPUT[Bob](w):

$w^B \leftarrow\$ \{0,1\}$

$w^A \leftarrow$ “next bit of y” $\oplus$ w^B

case XOR(w_0,w_1,w_2):

$w_2^A \leftarrow w_0^A \oplus w_1^A$; $w_2^B \leftarrow w_0^B \oplus w_1^B$

case AND(w_0,w_1,w_2):

$r \leftarrow\$ \{0,1\}$

$s \leftarrow\$ \{0,1\}$

$w_2^A \leftarrow r \oplus (s \oplus w_0^B \cdot w_1^A) \oplus w_0^A \cdot w_1^A$

$w_2^B \leftarrow s \oplus (r \oplus w_0^A \cdot w_1^B) \oplus w_0^B \cdot w_1^B$

case OUTPUT(w):

w^B

Sim$_{\text{Alice}}^{\text{GMW(C)}}$(x,C(x,y)):

for g in C; **switch** on g:

case INPUT[Alice](w):

$w^A \leftarrow\$ \{0,1\}$

case INPUT[Bob](w):

$w^A \leftarrow\$ \{0,1\}$

case XOR(w_0,w_1,w_2):

$w_2^A \leftarrow w_0^A \oplus w_1^A$

case AND(w_0,w_1,w_2):

$r \leftarrow\$ \{0,1\}$

$s \leftarrow\$ \{0,1\}$

$w_2^A \leftarrow r \oplus s \oplus w_0^A \cdot w_1^A$

case OUTPUT(w):

$w^B \leftarrow$ “next bit of C(x,y)” $\oplus$ w^A

denotes “add this to Alice's view”

View$_{\text{Alice}}^{\text{GMW(C)}}$(x,y):

for g in C; **switch** on g:

case INPUT[Alice](w):

$w^A \leftarrow\$ \{0,1\}$

$w^B \leftarrow$ “next bit of x” $\oplus$ w^A

case INPUT[Bob](w):

$w^B \leftarrow\$ \{0,1\}$

$w^A \leftarrow$ “next bit of y” $\oplus$ w^B

case XOR(w_0,w_1,w_2):

$w_2^A \leftarrow w_0^A \oplus w_1^A$; $w_2^B \leftarrow w_0^B \oplus w_1^B$

case AND(w_0,w_1,w_2):

$r \leftarrow\$ \{0,1\}$

$s \leftarrow\$ \{0,1\}$

$w_2^A \leftarrow r \oplus (s \oplus w_0^B \cdot w_1^A) \oplus w_0^A \cdot w_1^A$

$w_2^B \leftarrow s \oplus (r \oplus w_0^A \cdot w_1^B) \oplus w_0^B \cdot w_1^B$

case OUTPUT(w):

w^B

Sim$_{\text{Alice}}^{\text{GMW(C)}}$(x,C(x,y)):

for g in C; **switch** on g:

case INPUT[Alice](w):

$w^A \leftarrow\$ \{0,1\}$

case INPUT[Bob](w):

$w^A \leftarrow\$ \{0,1\}$

case XOR(w_0,w_1,w_2):

$w_2^A \leftarrow w_0^A \oplus w_1^A$

case AND(w_0,w_1,w_2):

$r \leftarrow\$ \{0,1\}$

$s \leftarrow\$ \{0,1\}$

$w_2^A \leftarrow r \oplus s \oplus w_0^A \cdot w_1^A$

case OUTPUT(w):

$w^B \leftarrow$ “next bit of C(x,y)” $\oplus$ w^A

denotes “add this to Alice's view”

View$_{\text{Alice}}^{\text{GMW(C)}}$(x,y):

for g in C; **switch** on g:

case INPUT[Alice](w):

$w^A \leftarrow\$ \{0,1\}$

$w^B \leftarrow$ “next bit of x” $\oplus$ w^A

case INPUT[Bob](w):

$w^B \leftarrow\$ \{0,1\}$

$w^A \leftarrow$ “next bit of y” $\oplus$ w^B

case XOR(w_0,w_1,w_2):

$w_2^A \leftarrow w_0^A \oplus w_1^A$; $w_2^B \leftarrow w_0^B \oplus w_1^B$

case AND(w_0,w_1,w_2):

$r \leftarrow\$ \{0,1\}$

$s \leftarrow\$ \{0,1\}$

$w_2^A \leftarrow r \oplus (s \oplus w_0^B \cdot w_1^A) \oplus w_0^A \cdot w_1^A$

$w_2^B \leftarrow s \oplus (r \oplus w_0^A \cdot w_1^B) \oplus w_0^B \cdot w_1^B$

case OUTPUT(w):

w^B

Sent by Trusted Third Party

Sim$_{\text{Alice}}^{\text{GMW(C)}}$(x,C(x,y)):

for g in C; **switch** on g:

case INPUT[Alice](w):

$w^A \leftarrow\$ \{0,1\}$

case INPUT[Bob](w):

$w^A \leftarrow\$ \{0,1\}$

case XOR(w_0,w_1,w_2):

$w_2^A \leftarrow w_0^A \oplus w_1^A$

case AND(w_0,w_1,w_2):

$r \leftarrow\$ \{0,1\}$

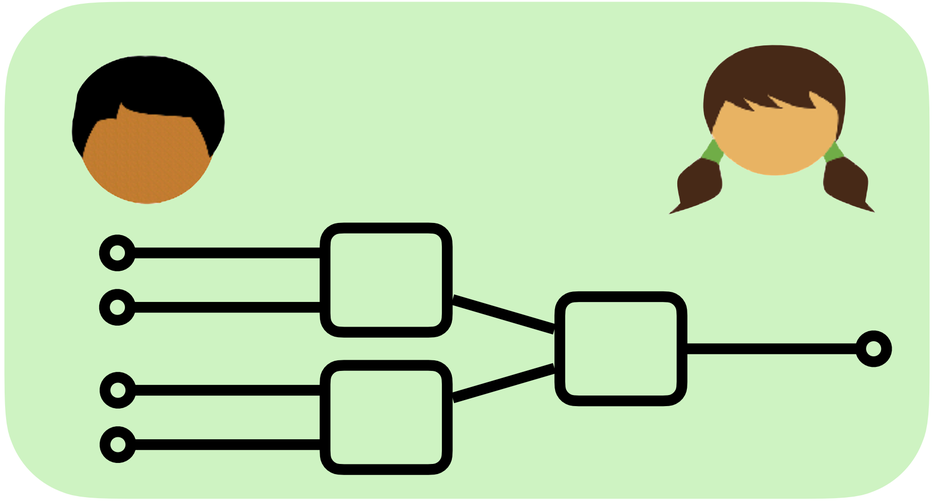
$s \leftarrow\$ \{0,1\}$

$w_2^A \leftarrow r \oplus s \oplus w_0^A \cdot w_1^A$

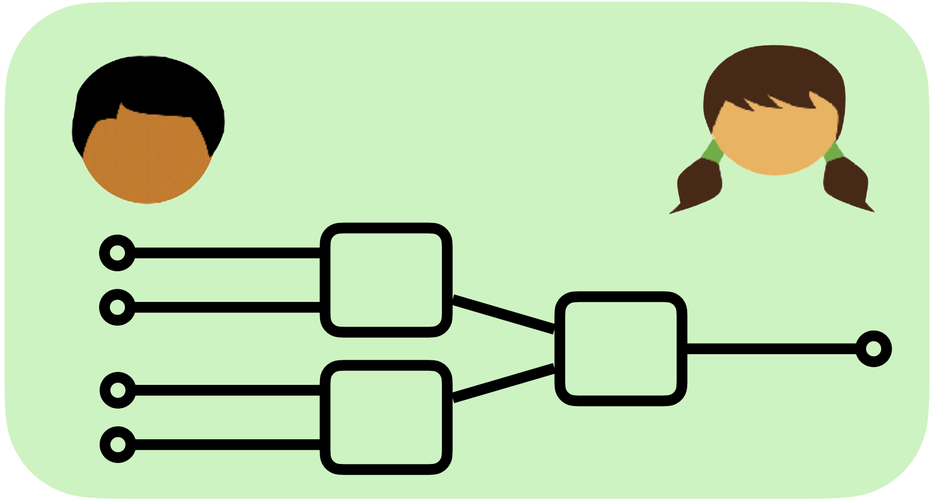
case OUTPUT(w):

$w^B \leftarrow$ “next bit of C(x,y)” $\oplus$ w^A

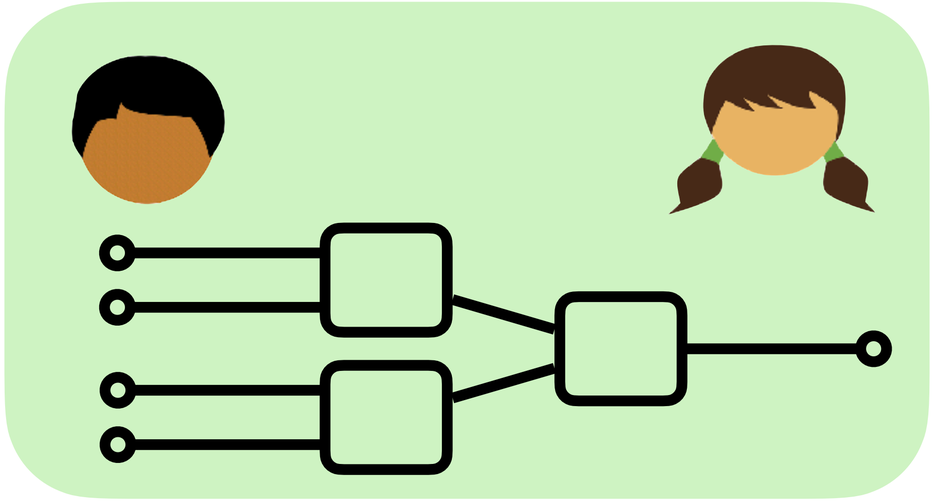
denotes “add this to Alice's view”



In GMW, Number of protocol rounds scales with multiplicative depth of C

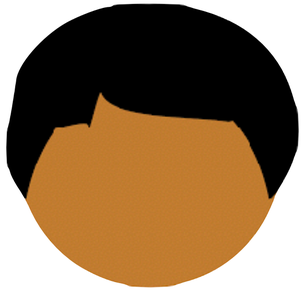


In GMW, Number of protocol rounds scales with multiplicative depth of C



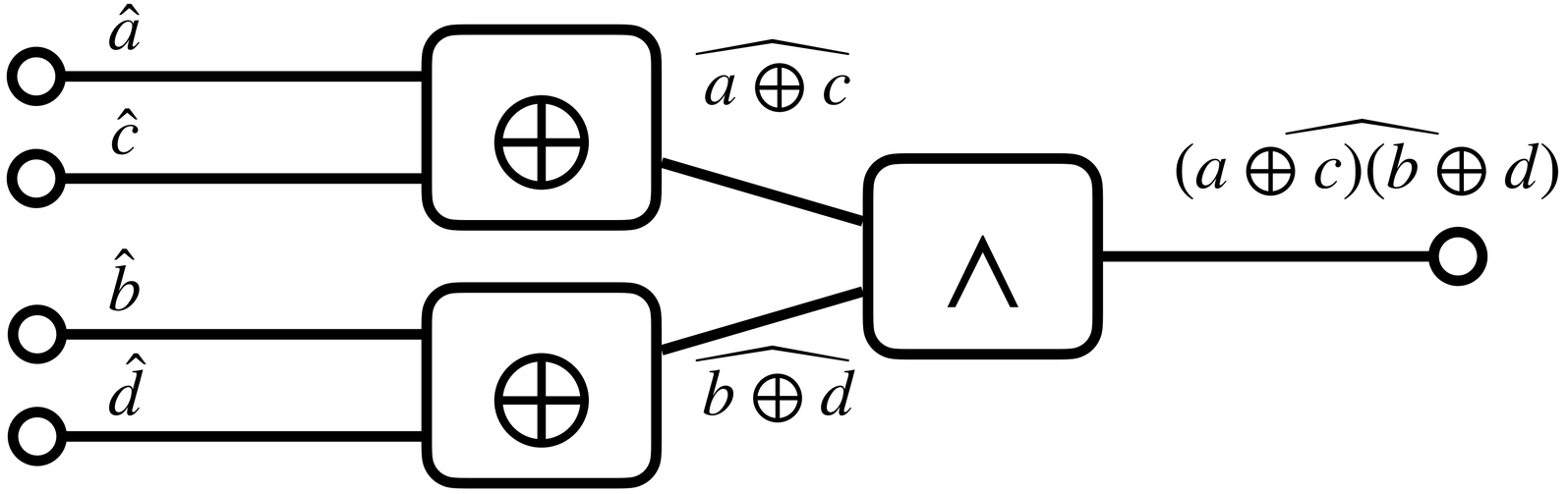
In GMW, Number of protocol rounds scales with multiplicative depth of C

Our protocol's efficiency is fundamentally bounded by the speed of light



a, b

c, d

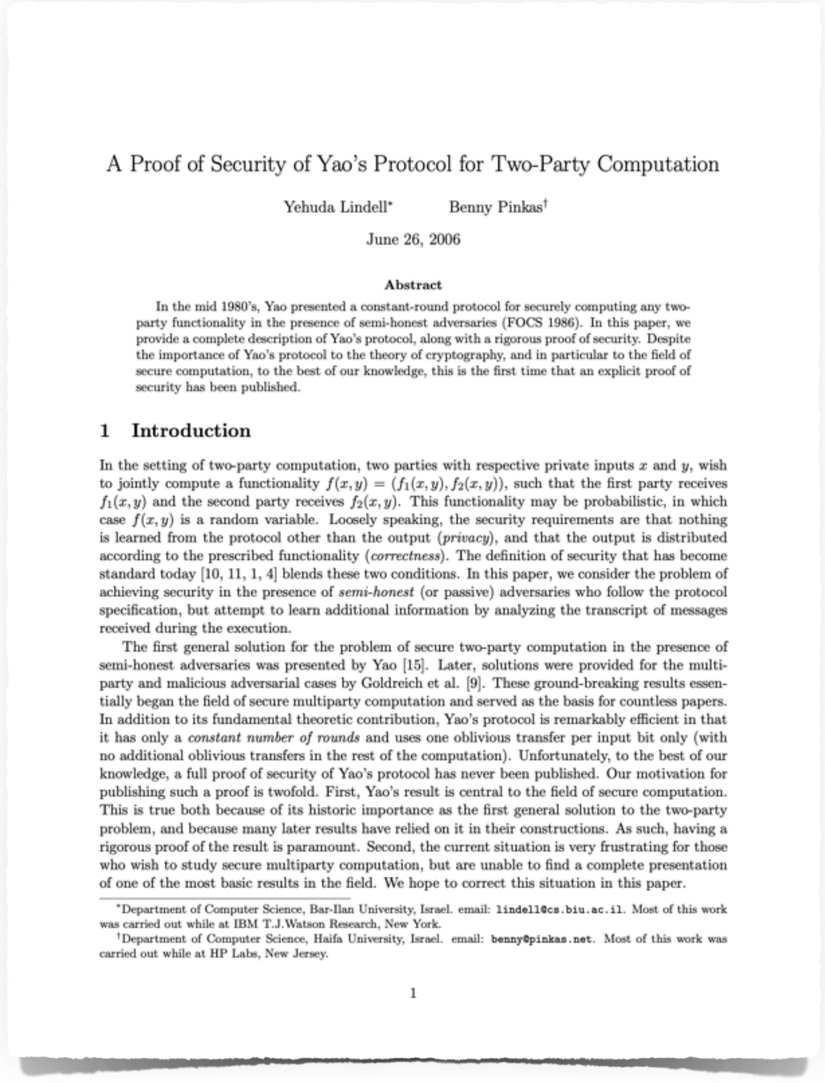

$\hat{a}$
$\hat{c}$
$\hat{b}$
$\hat{d}$
$\widehat{a \oplus c}$
$\widehat{b \oplus d}$
$\widehat{(a \oplus c)(b \oplus d)}$

Garbled Circuits (GC)

Fundamental approach to MPC

Allows constant round protocols for arbitrary programs

[Andrew Yao, 1986, FOCS Conference Talk]



A Proof of Security of Yao's Protocol for Two-Party Computation

Yehuda Lindell* Benny Pinkas†

June 26, 2006

Abstract

In the mid 1980's, Yao presented a constant-round protocol for securely computing any two-party functionality in the presence of semi-honest adversaries (FOCS 1986). In this paper, we provide a complete description of Yao's protocol, along with a rigorous proof of security. Despite the importance of Yao's protocol to the theory of cryptography, and in particular to the field of secure computation, to the best of our knowledge, this is the first time that an explicit proof of security has been published.

1 Introduction

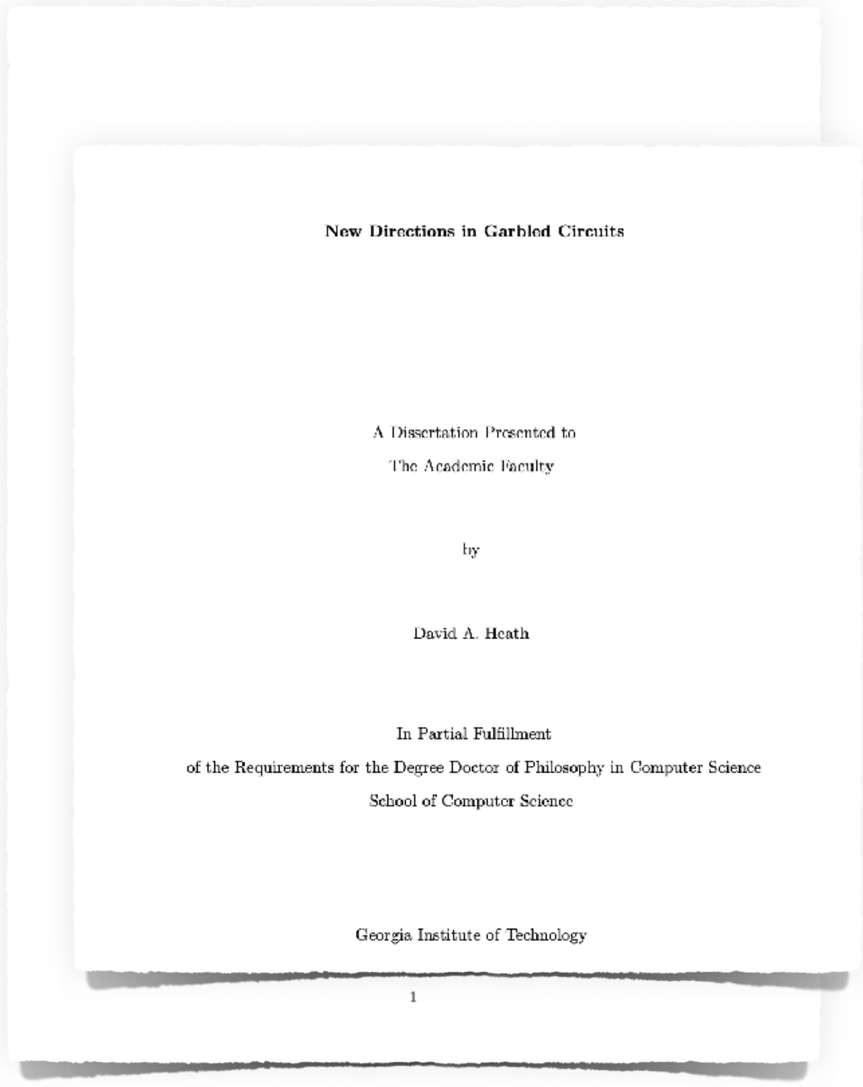
In the setting of two-party computation, two parties with respective private inputs x and y, wish to jointly compute a functionality $f(x,y) = (f_1(x,y), f_2(x,y))$, such that the first party receives $f_1(x,y)$ and the second party receives $f_2(x,y)$. This functionality may be probabilistic, in which case $f(x,y)$ is a random variable. Loosely speaking, the security requirements are that nothing is learned from the protocol other than the output (*privacy*), and that the output is distributed according to the prescribed functionality (*correctness*). The definition of security that has become standard today [10, 11, 1, 4] blends these two conditions. In this paper, we consider the problem of achieving security in the presence of *semi-honest* (or passive) adversaries who follow the protocol specification, but attempt to learn additional information by analyzing the transcript of messages received during the execution.

The first general solution for the problem of secure two-party computation in the presence of semi-honest adversaries was presented by Yao [15]. Later, solutions were provided for the multi-party and malicious adversarial cases by Goldreich et al. [9]. These ground-breaking results essentially began the field of secure multiparty computation and served as the basis for countless papers. In addition to its fundamental theoretic contribution, Yao's protocol is remarkably efficient in that it has only a *constant number of rounds* and uses one oblivious transfer per input bit only (with no additional oblivious transfers in the rest of the computation). Unfortunately, to the best of our knowledge, a full proof of security of Yao's protocol has never been published. Our motivation for publishing such a proof is twofold. First, Yao's result is central to the field of secure computation. This is true both because of its historic importance as the first general solution to the two-party problem, and because many later results have relied on it in their constructions. As such, having a rigorous proof of the result is paramount. Second, the current situation is very frustrating for those who wish to study secure multiparty computation, but are unable to find a complete presentation of one of the most basic results in the field. We hope to correct this situation in this paper.

*Department of Computer Science, Bar-Ilan University, Israel. email: lindell@cs.biu.ac.il. Most of this work was carried out while at IBM T.J.Watson Research, New York.

†Department of Computer Science, Haifa University, Israel. email: benny@pinkas.net. Most of this work was carried out while at HP Labs, New Jersey.

[Andrew Yao, 1986, FOCS Conference Talk]



New Directions in Garbled Circuits

A Dissertation Presented to
The Academic Faculty

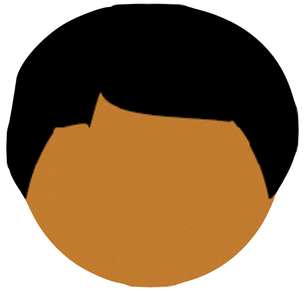
by

David A. Heath

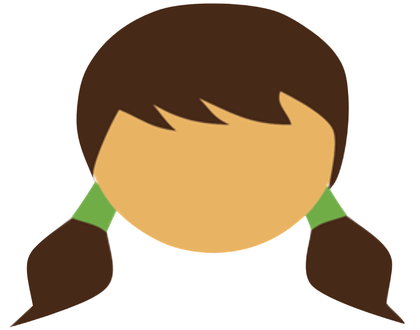
In Partial Fulfillment
of the Requirements for the Degree Doctor of Philosophy in Computer Science
School of Computer Science

Georgia Institute of Technology

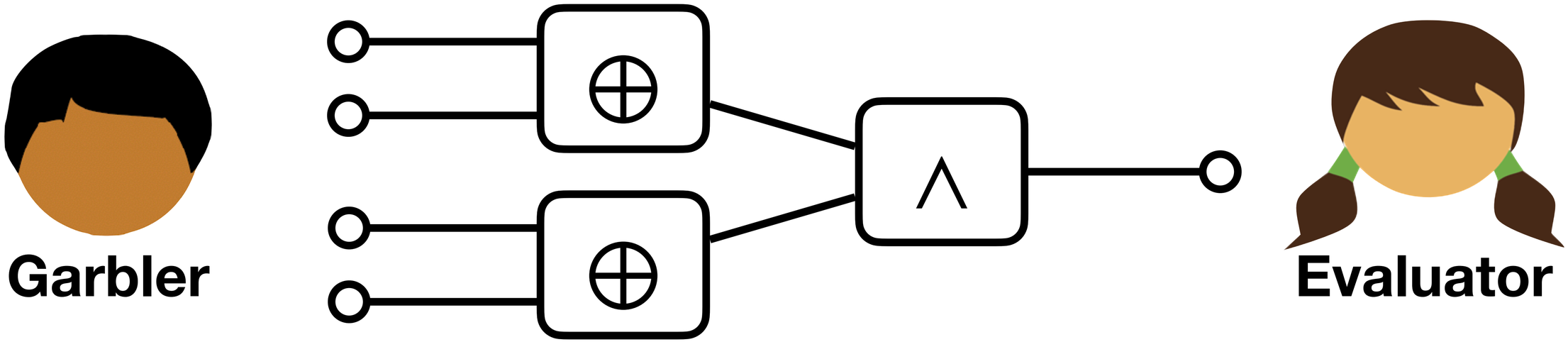
1

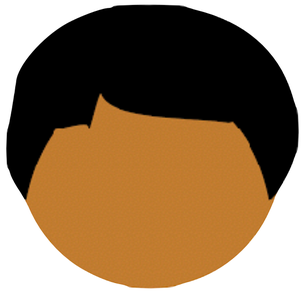


Garbler

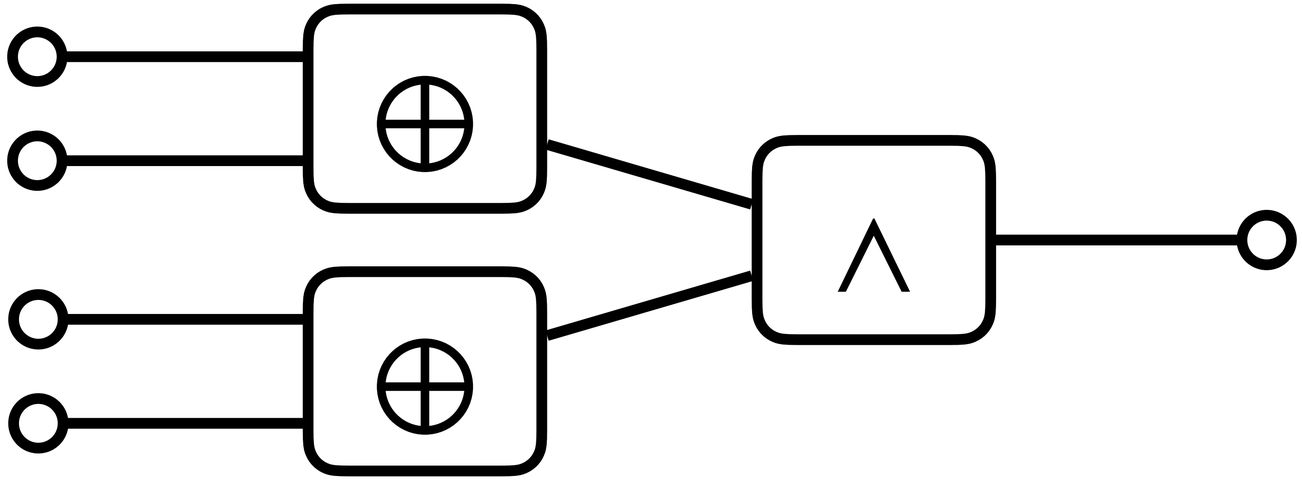


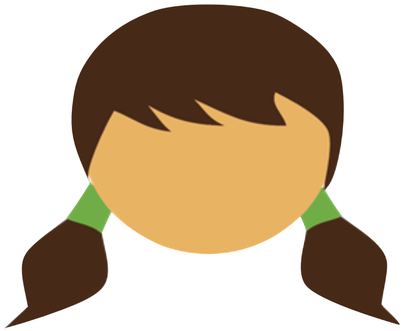
Evaluator


Garbler
Evaluator

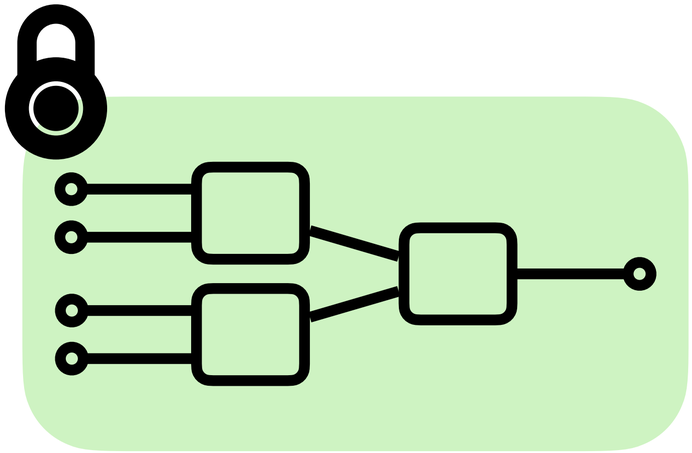


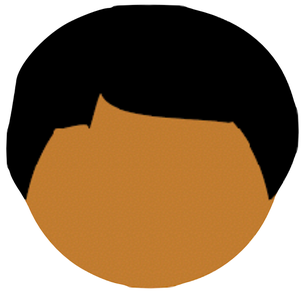
Garbler



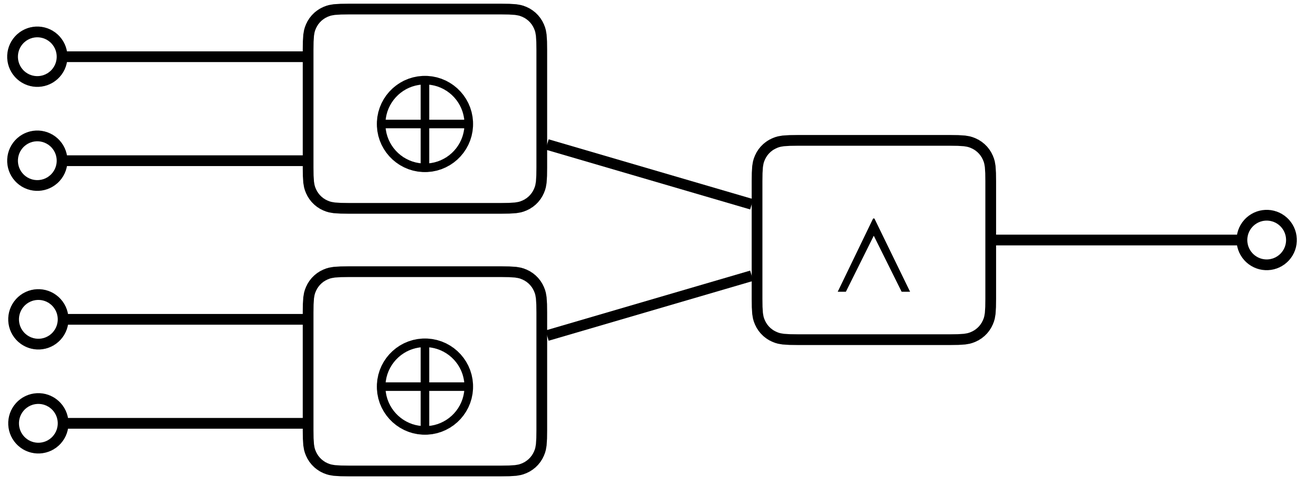


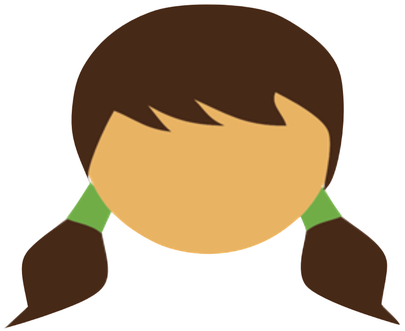
Evaluator



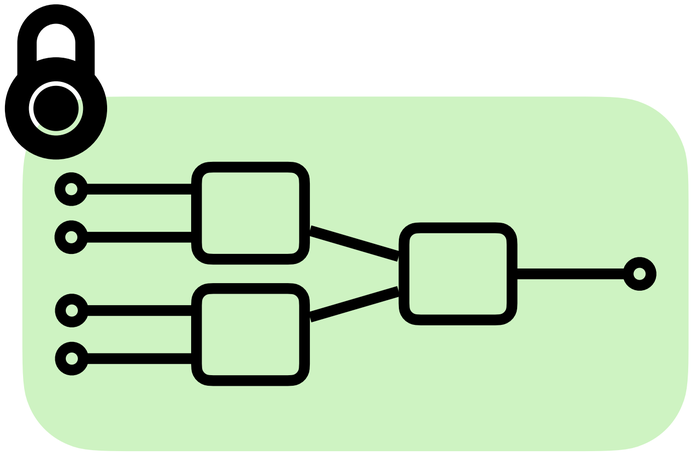


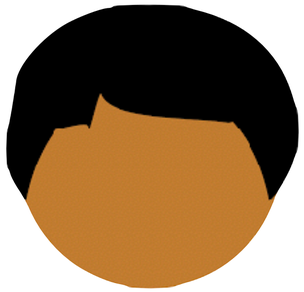
Garbler



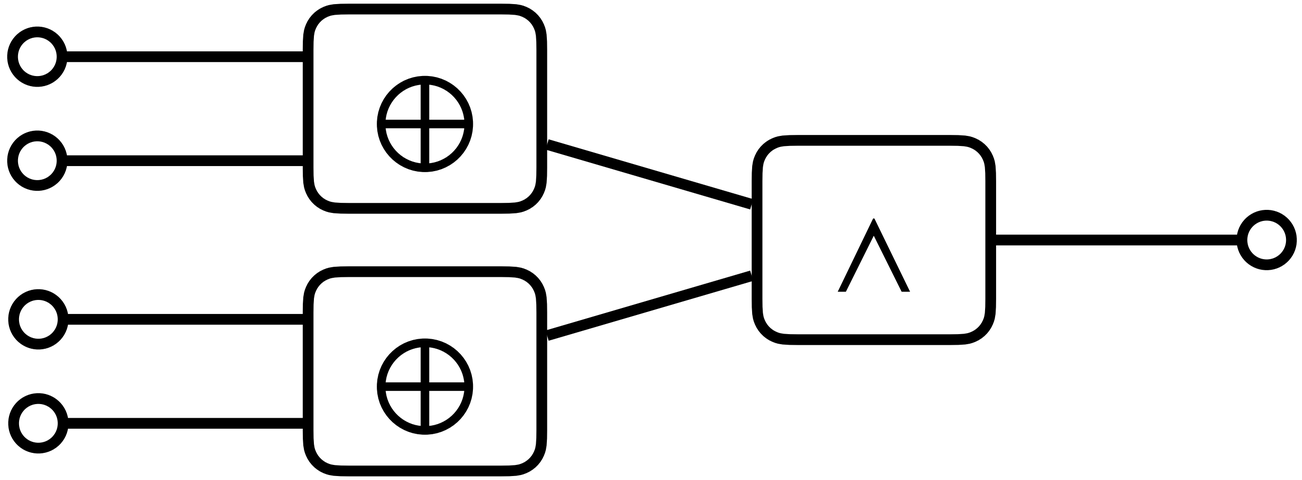


Evaluator



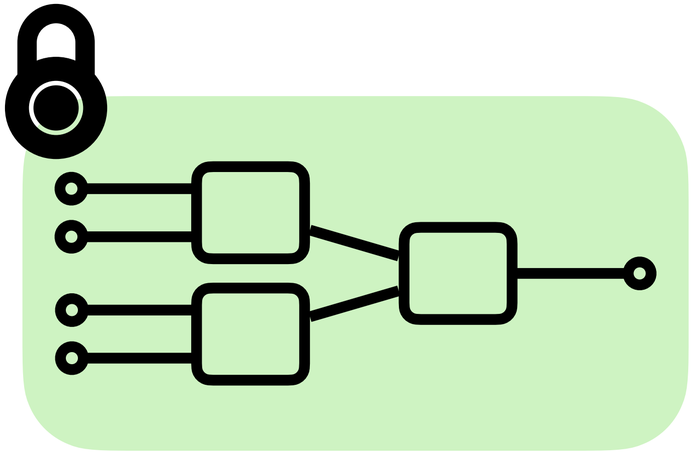


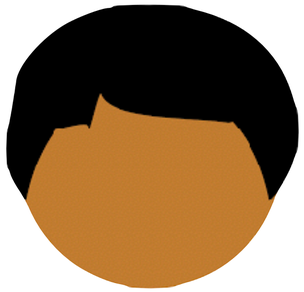
Garbler



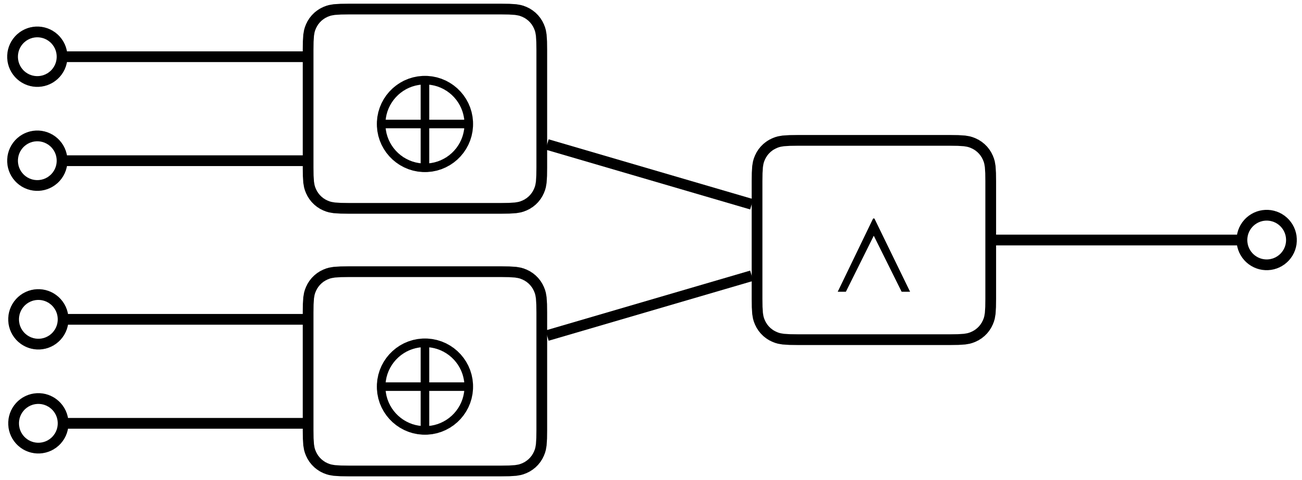
Evaluator

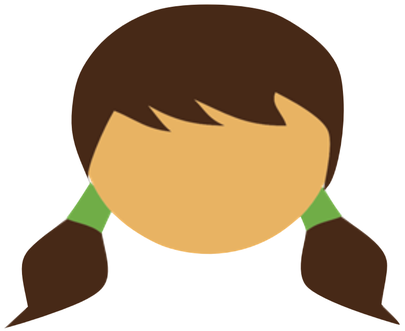
(OT)



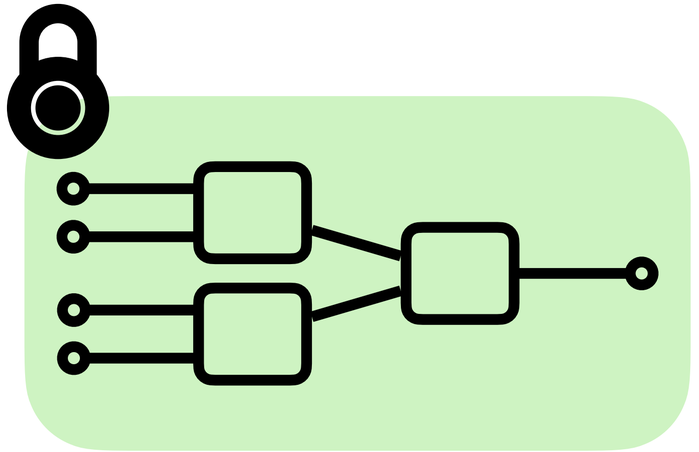


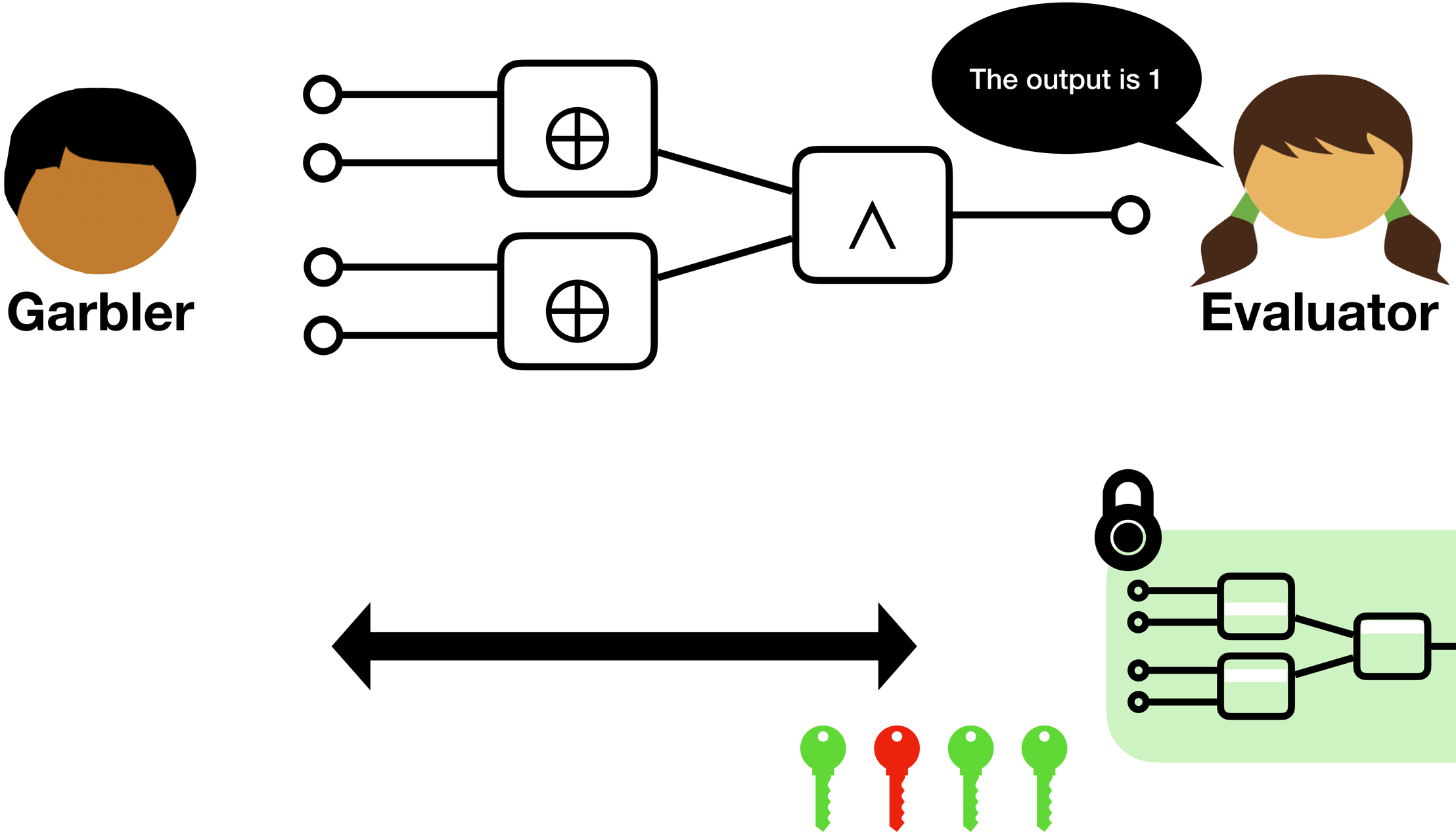
Garbler

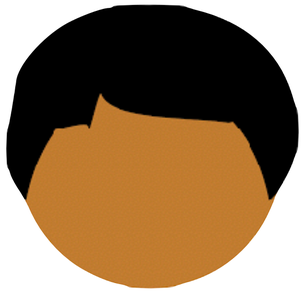




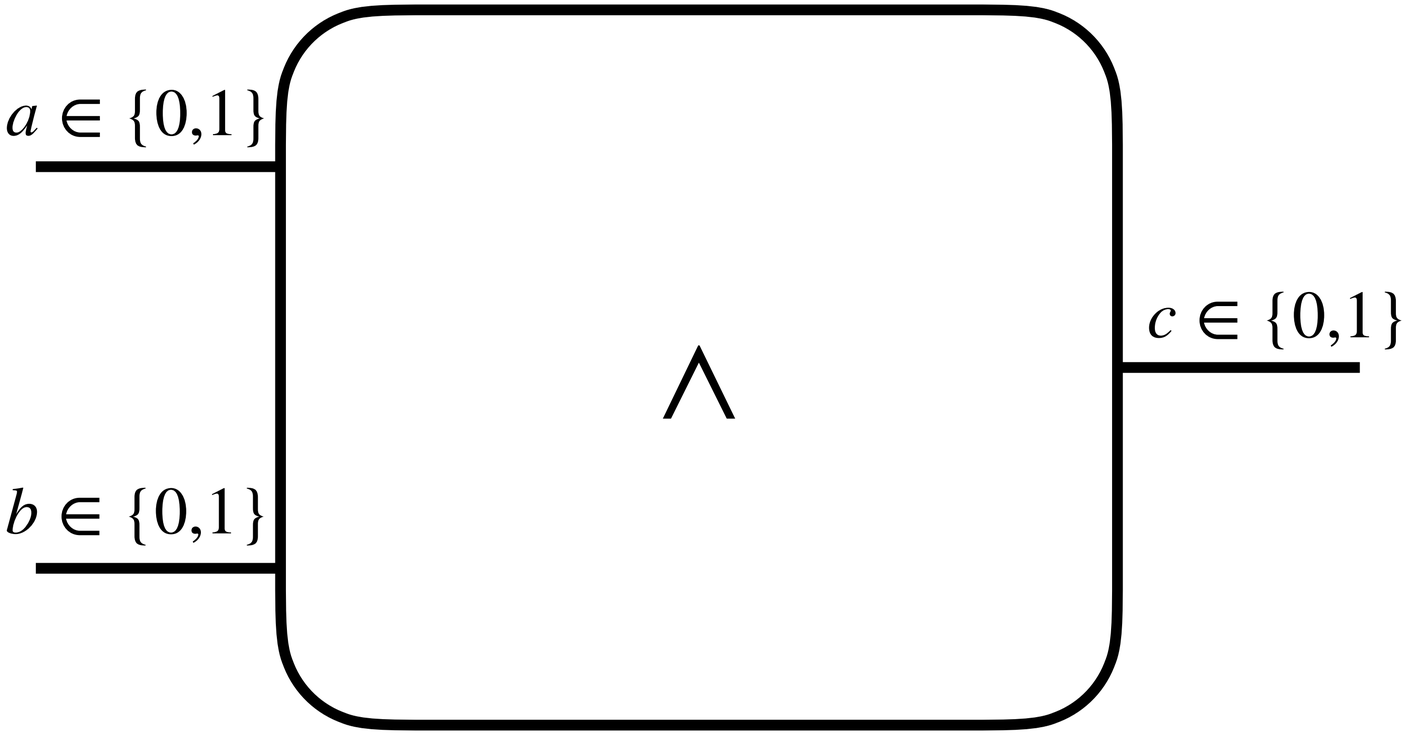
Evaluator

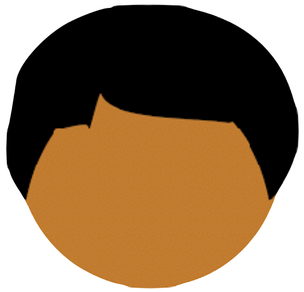



Garbler
The output is 1
Evaluator

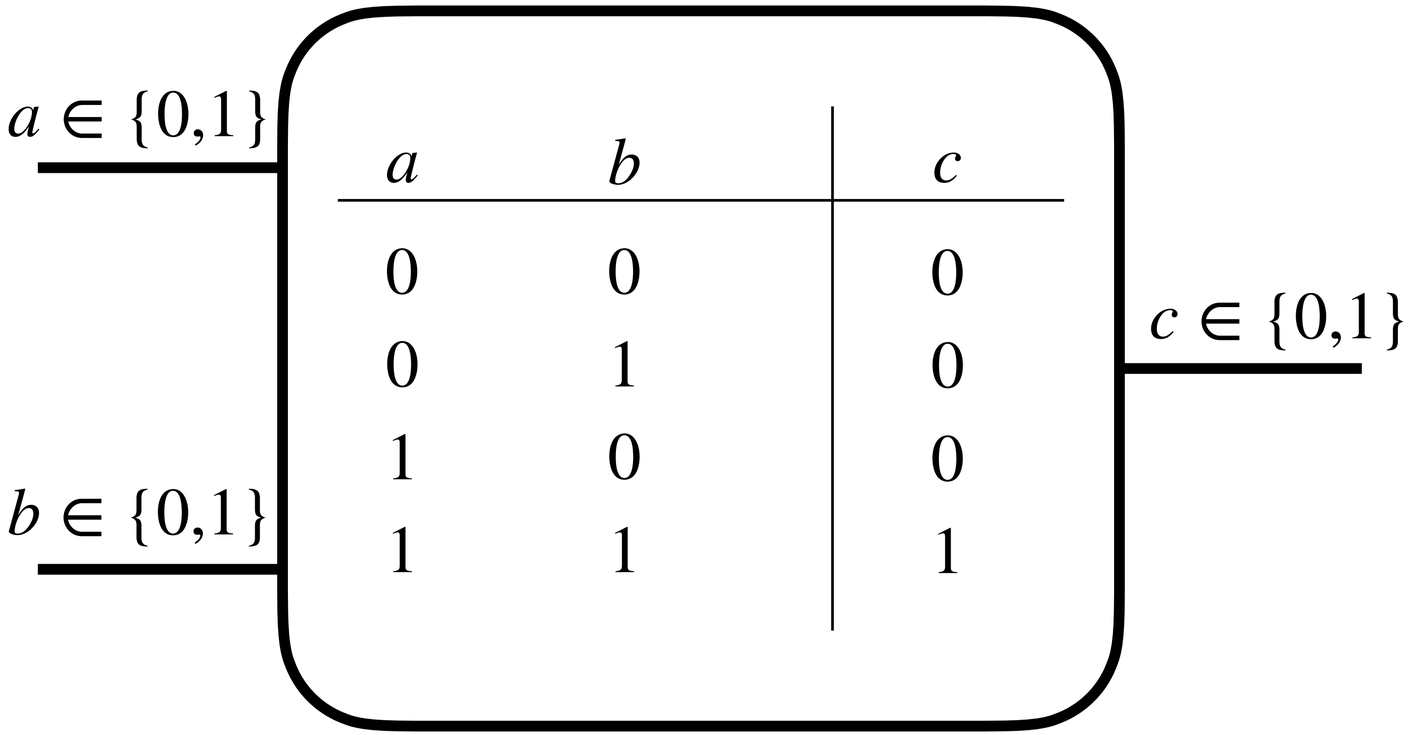


G

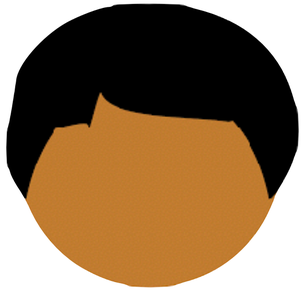




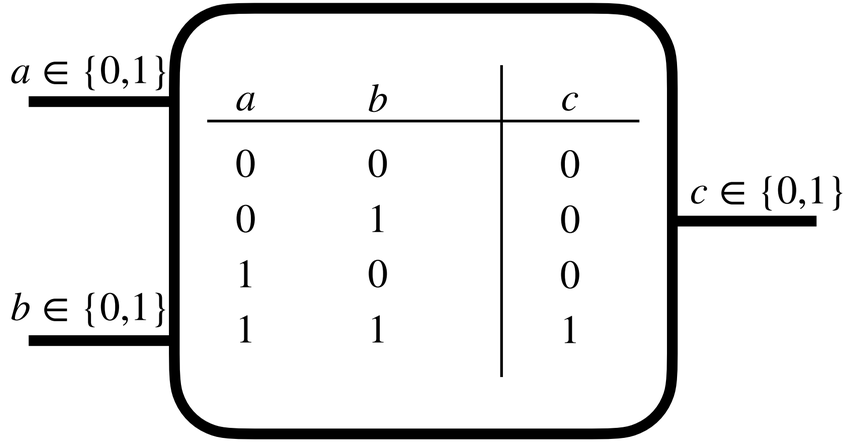
G



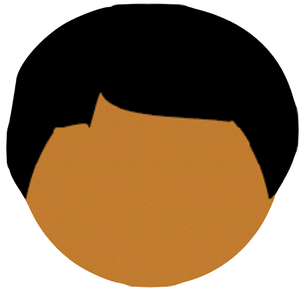
a	b	c
0	0	0
0	1	0
1	0	0
1	1	1



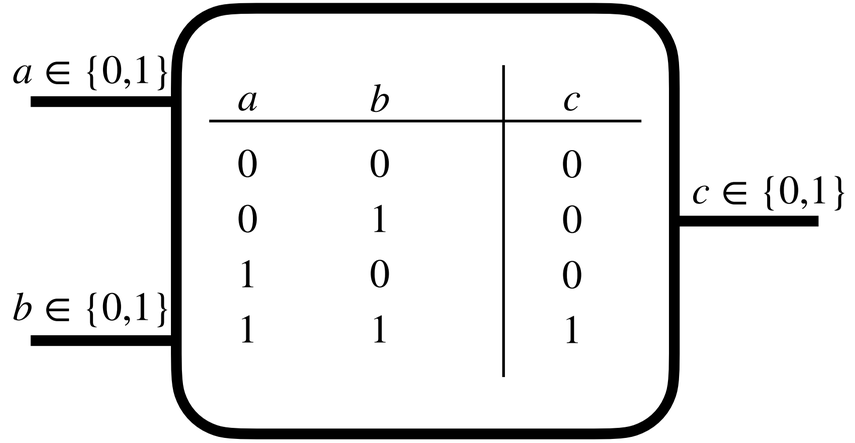
G

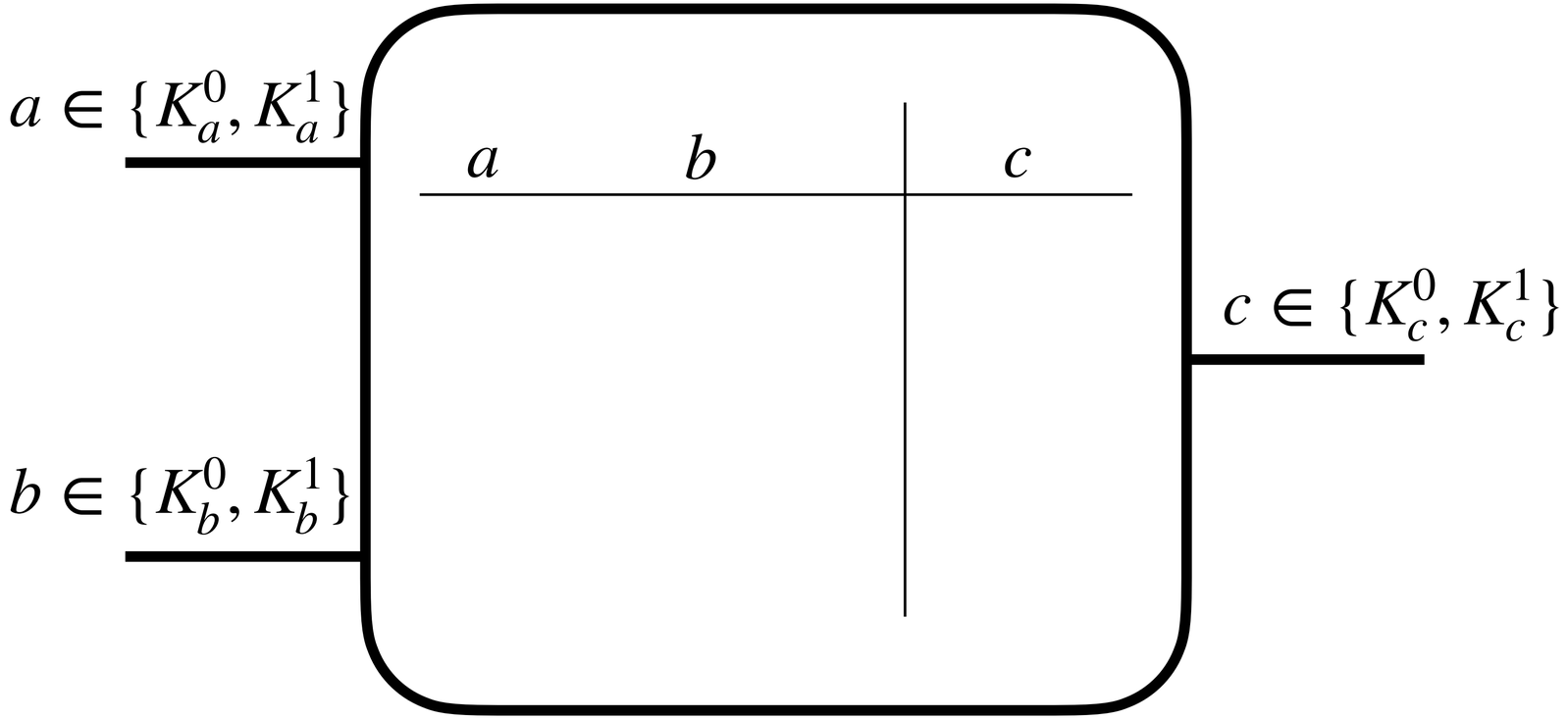


a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

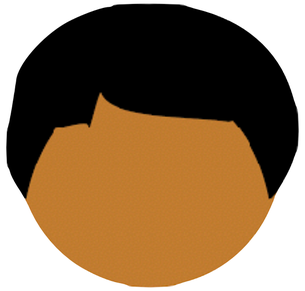


G

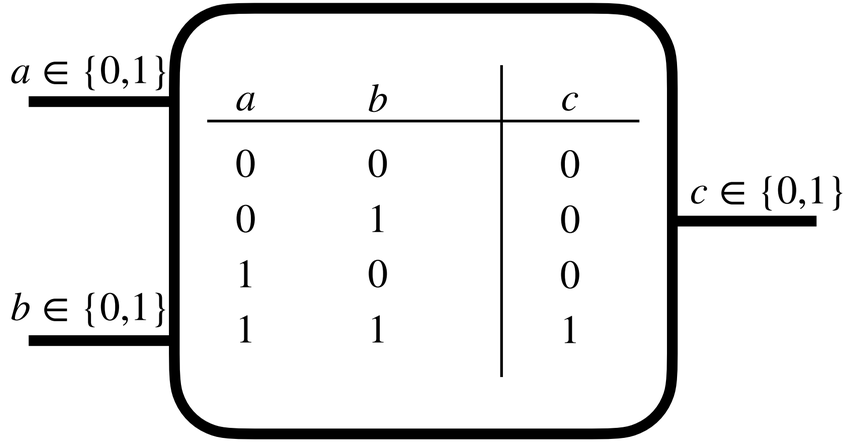


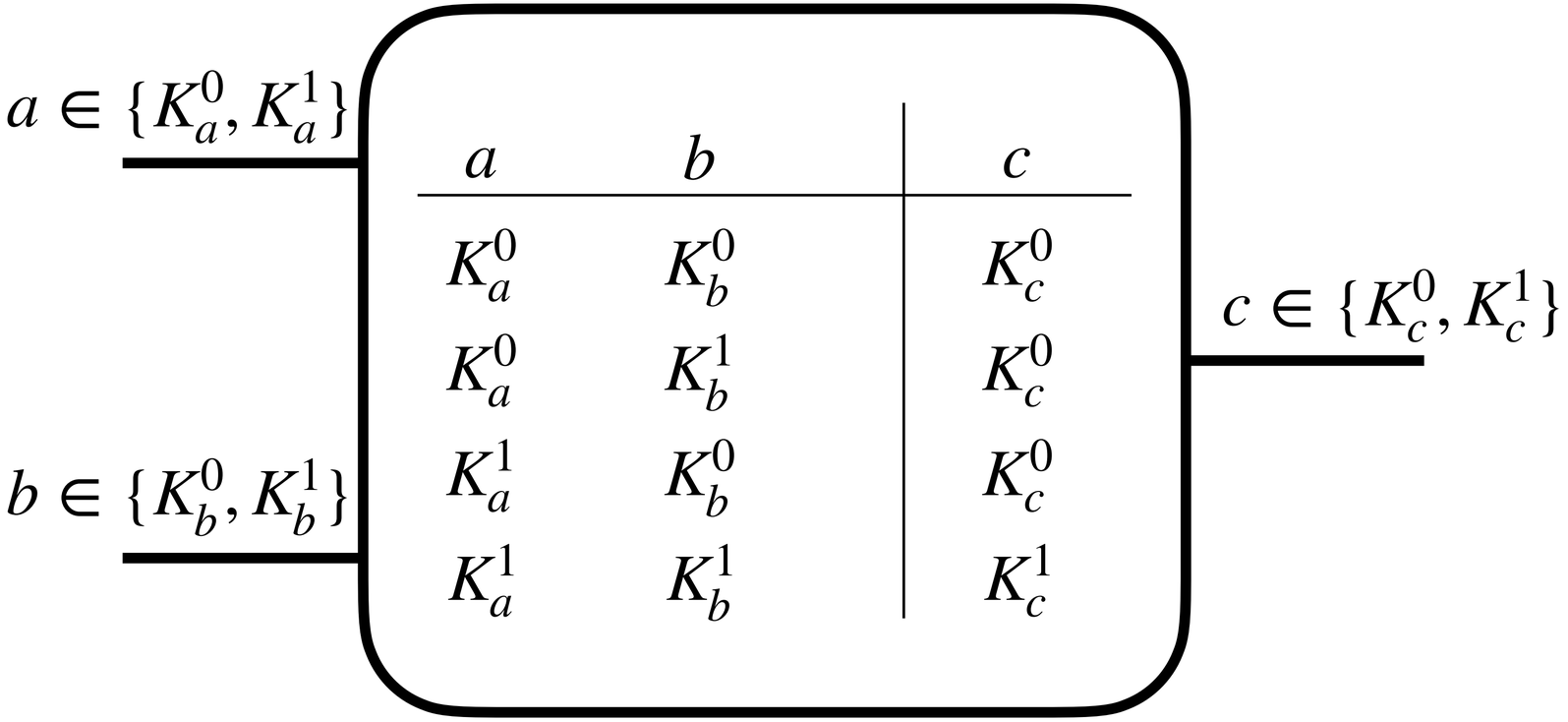


Each value K_j^i is a random *encryption key*

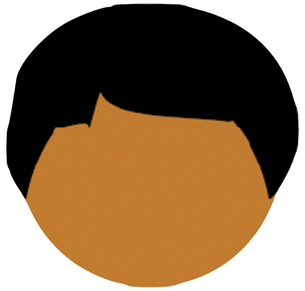


G

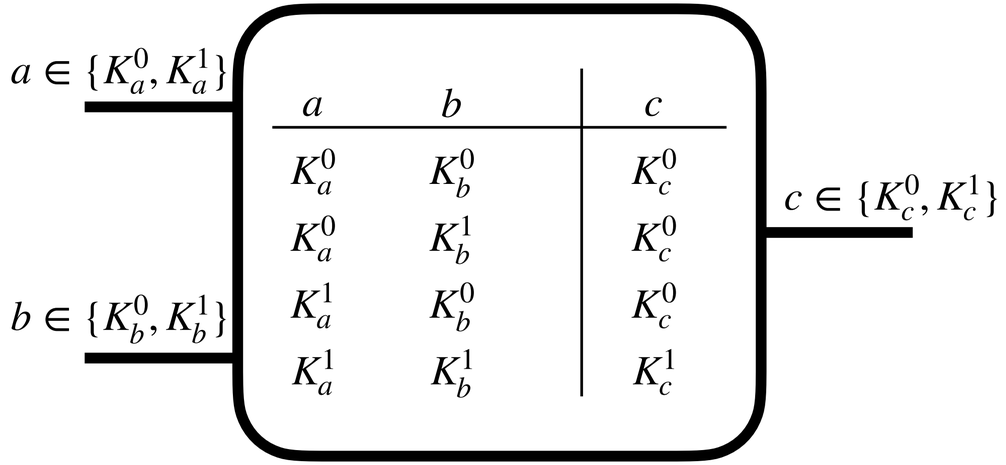




Each value K_j^i is a random *encryption key*

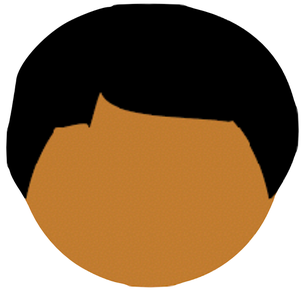


G

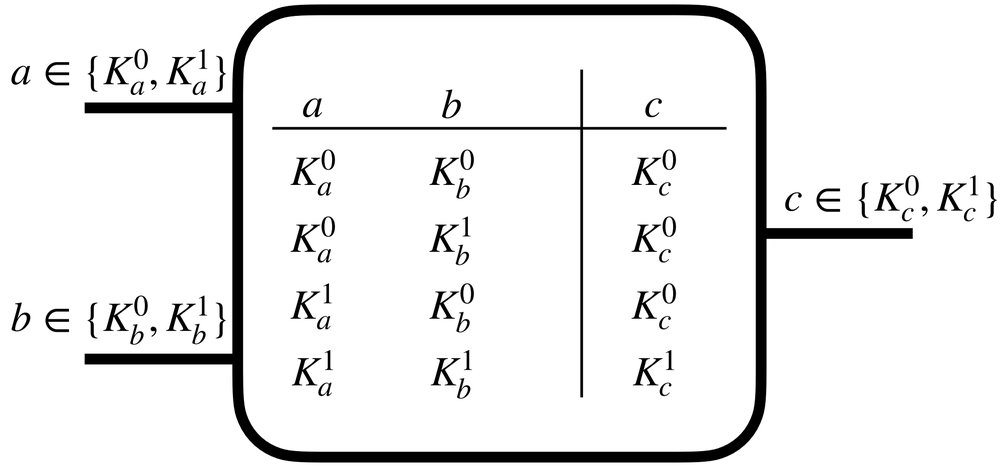


$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$

Each value K_j^i is a random *encryption key*



G

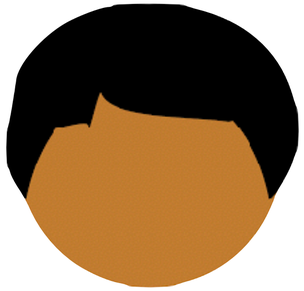


$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$

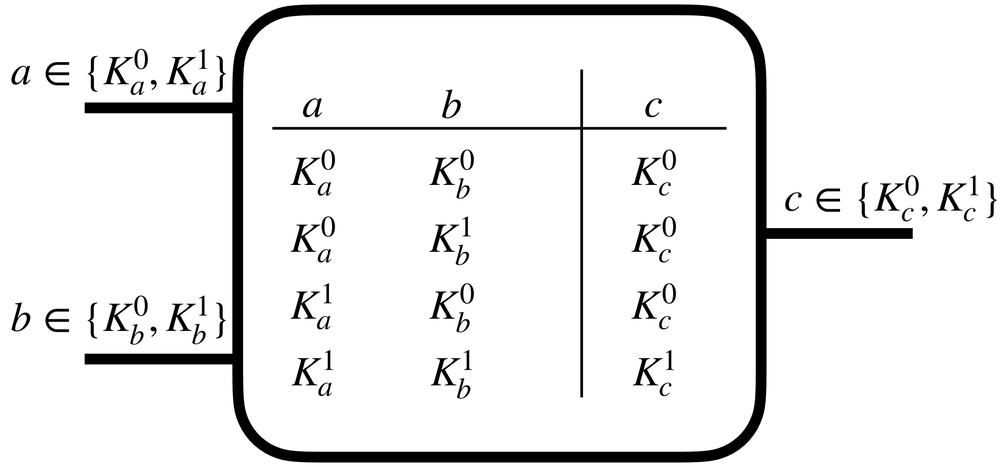
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$

$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$

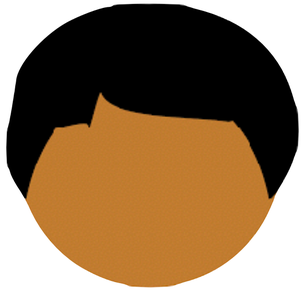
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$



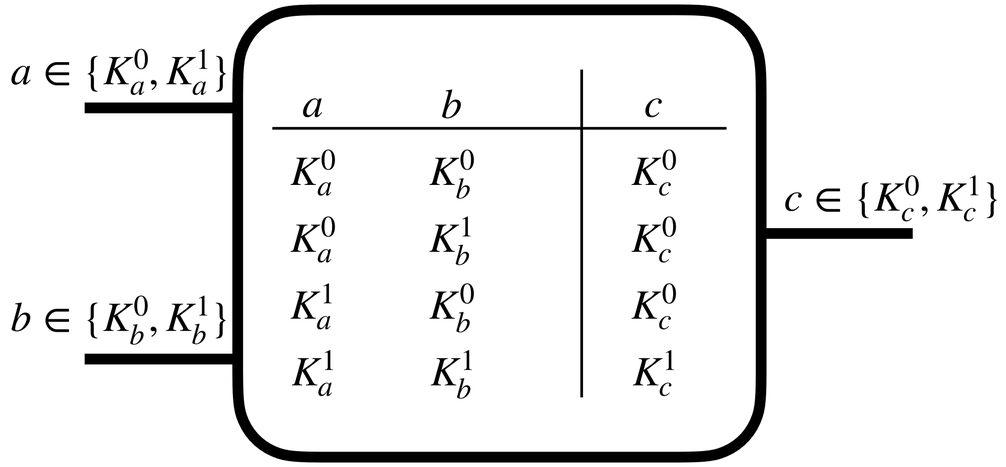
G



$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$



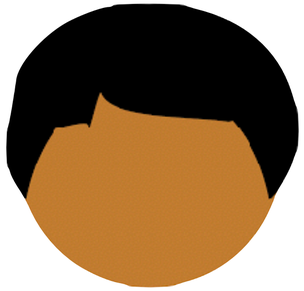
G



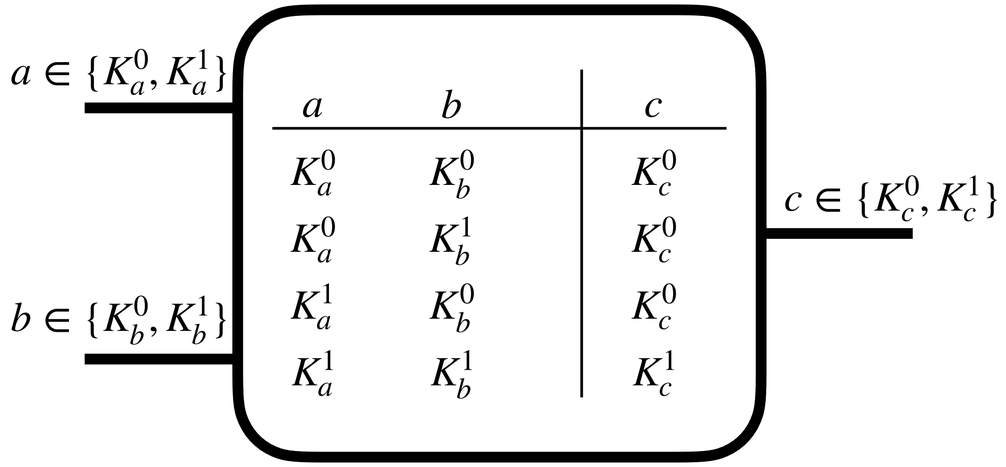
Observations:

If you have K_a^x, K_b^y, you can decrypt $K_a^{x \cdot y}$

$$\text{Enc}(K_a^0, \text{Enc}(K_b^0, K_c^0))$$
$$\text{Enc}(K_a^0, \text{Enc}(K_b^1, K_c^0))$$
$$\text{Enc}(K_a^1, \text{Enc}(K_b^0, K_c^0))$$
$$\text{Enc}(K_a^1, \text{Enc}(K_b^1, K_c^1))$$



G



Observations:

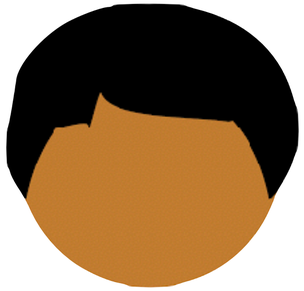
If you have K_a^x, K_b^y, you can decrypt $K_a^{x \cdot y}$

$\text{Enc}(K_a^0, \text{Enc}(K_b^0, K_c^0))$

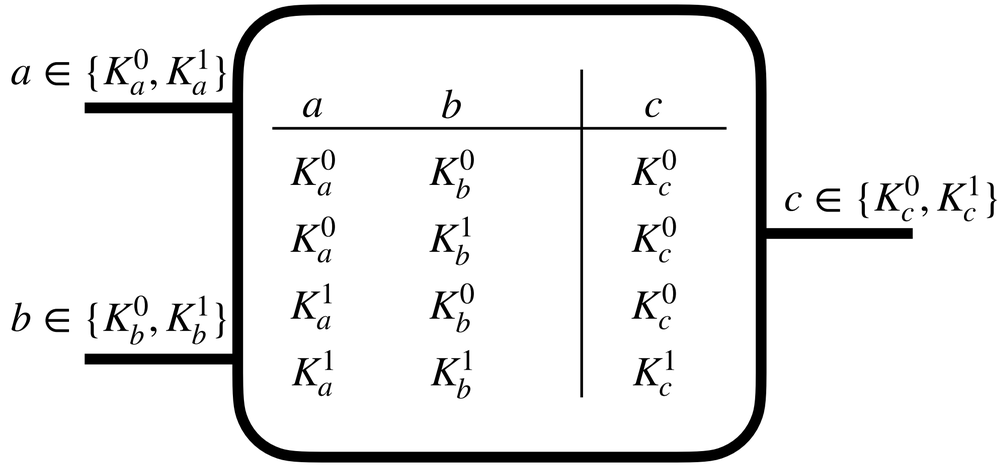
$\text{Enc}(K_a^0, \text{Enc}(K_b^1, K_c^0))$

$\text{Enc}(K_a^1, \text{Enc}(K_b^0, K_c^0))$

$\text{Enc}(K_a^1, \text{Enc}(K_b^1, K_c^1))$



G



Observations:

If you have K_a^x, K_b^y, you can decrypt $K_a^{x \cdot y}$

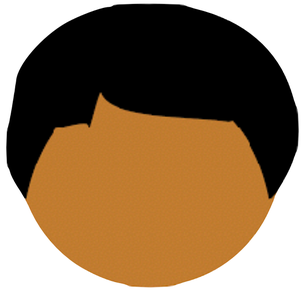
You cannot decrypt any row for which you are missing at least one key

$$\text{Enc}(K_a^0, \text{Enc}(K_b^0, K_c^0))$$

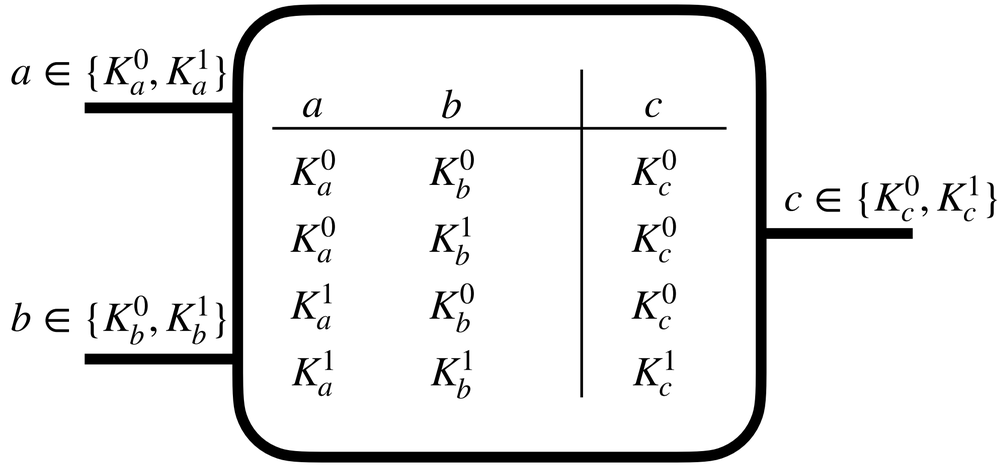
$$\text{Enc}(K_a^0, \text{Enc}(K_b^1, K_c^0))$$

$$\text{Enc}(K_a^1, \text{Enc}(K_b^0, K_c^0))$$

$$\text{Enc}(K_a^1, \text{Enc}(K_b^1, K_c^1))$$



G



Observations:

If you have K_a^x, K_b^y, you can decrypt $K_a^{x \cdot y}$

You cannot decrypt any row for which you are missing at least one key

Each key is random, so zero keys look like one keys

$$\text{Enc}(K_a^0, \text{Enc}(K_b^0, K_c^0))$$

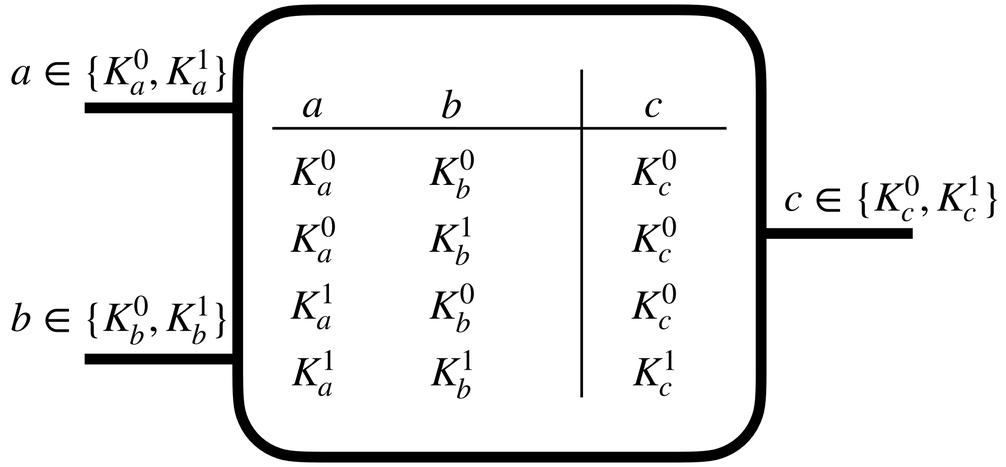
$$\text{Enc}(K_a^0, \text{Enc}(K_b^1, K_c^0))$$

$$\text{Enc}(K_a^1, \text{Enc}(K_b^0, K_c^0))$$

$$\text{Enc}(K_a^1, \text{Enc}(K_b^1, K_c^1))$$

Basic idea:

G chooses two keys per wire

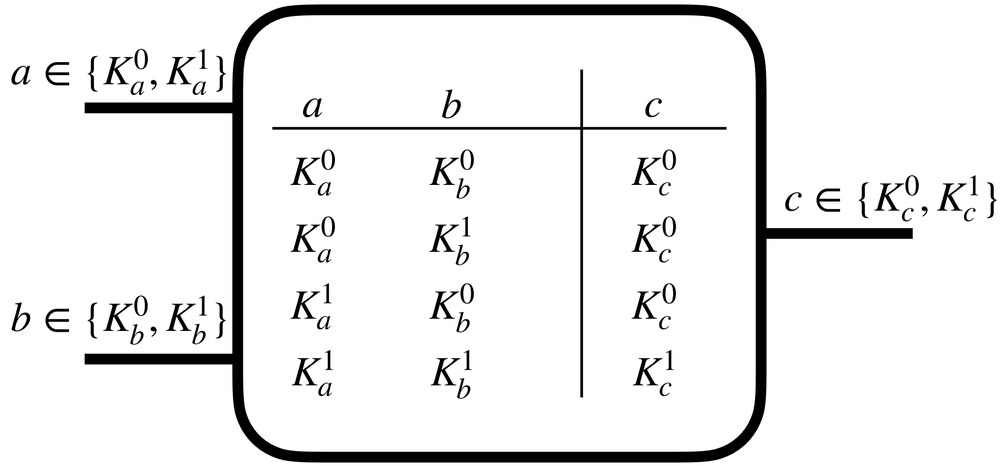


$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

Basic idea:

G chooses two keys per wire

G encrypts gate output keys according to gate input keys



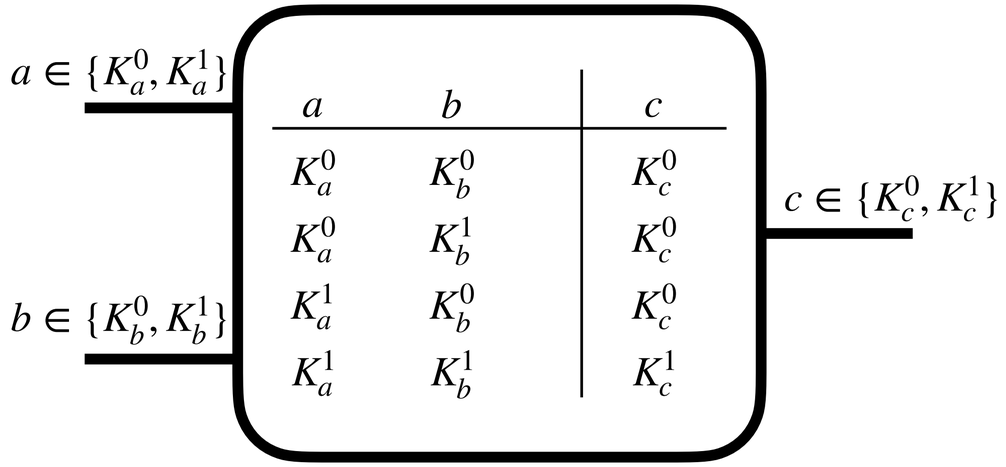
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

Basic idea:

G chooses two keys per wire

G encrypts gate output keys according to gate input keys

E receives circuit input keys corresponding to the party inputs



$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

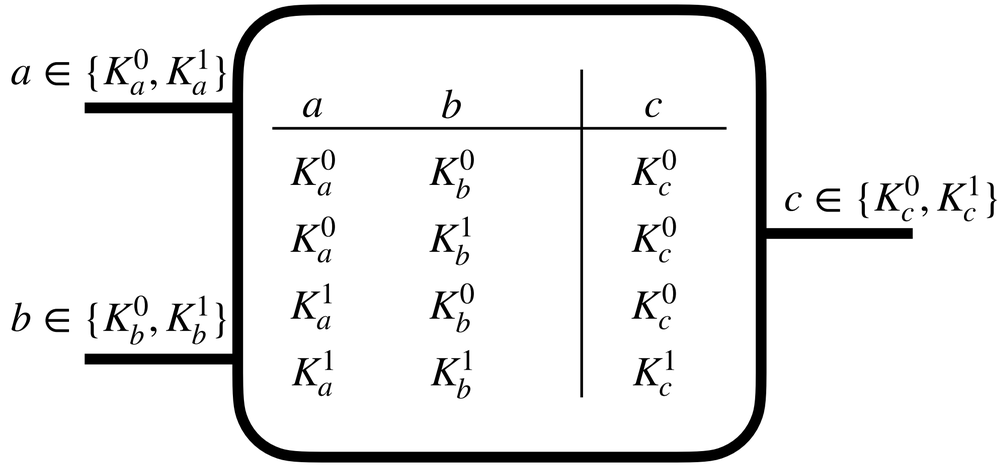
Basic idea:

G chooses two keys per wire

G encrypts gate output keys according to gate input keys

E receives circuit input keys corresponding to the party inputs

E decrypts each gate, eventually getting output keys, which can be jointly decoded



$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

Basic idea:

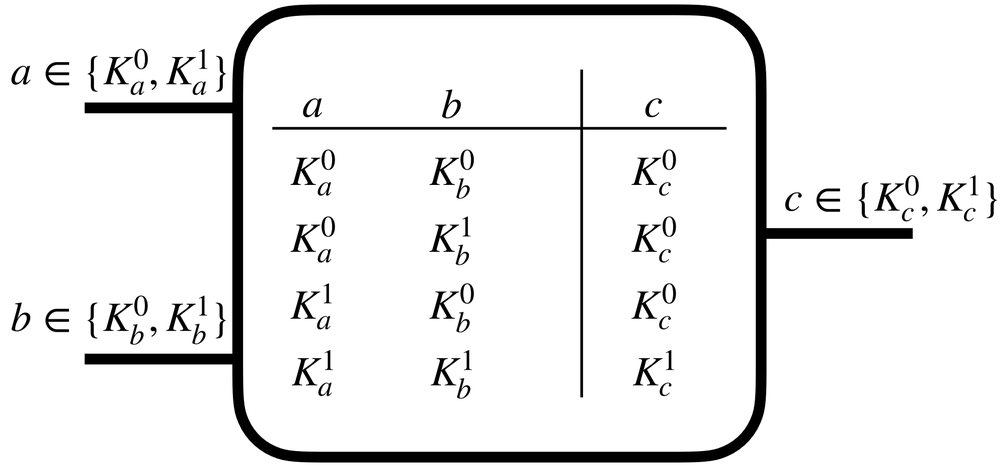
G chooses two keys per wire

G encrypts gate output keys according to gate input keys

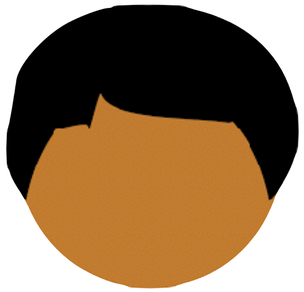
E receives circuit input keys corresponding to the party inputs

E decrypts each gate, eventually getting output keys, which can be jointly decoded

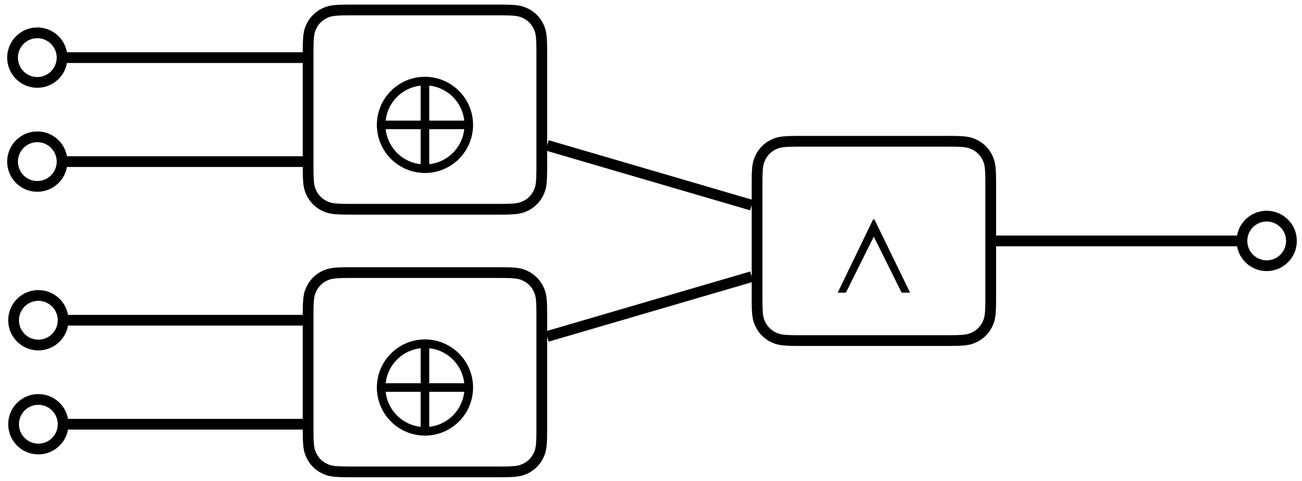
It is crucial that E only learn one key per wire

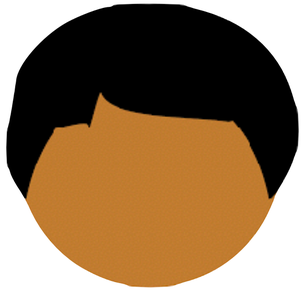


$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

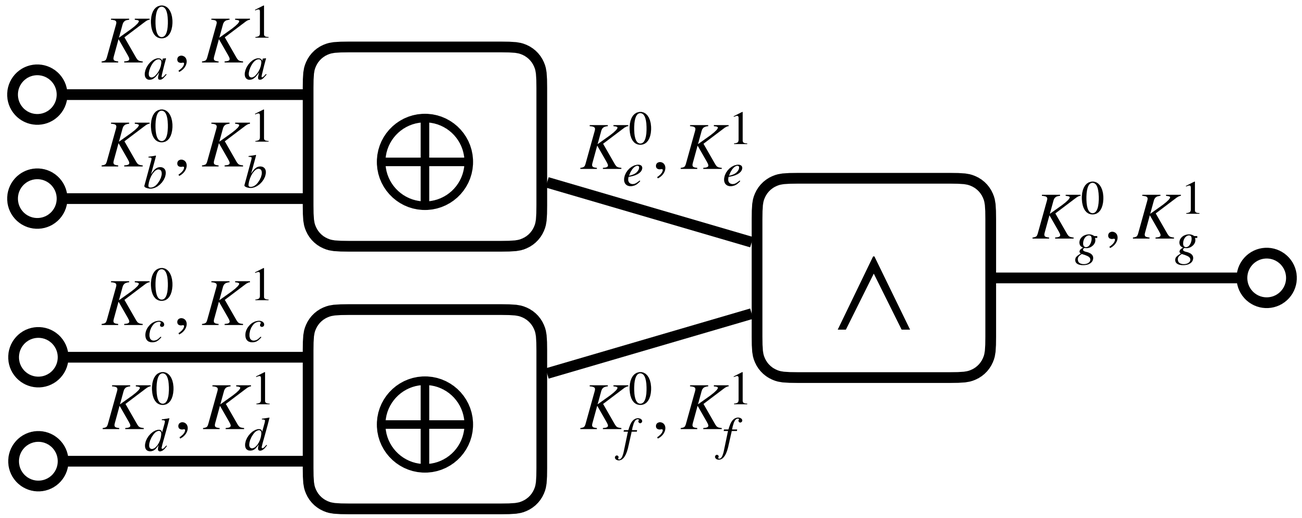


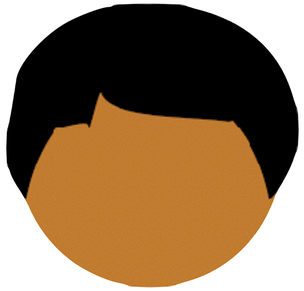
G



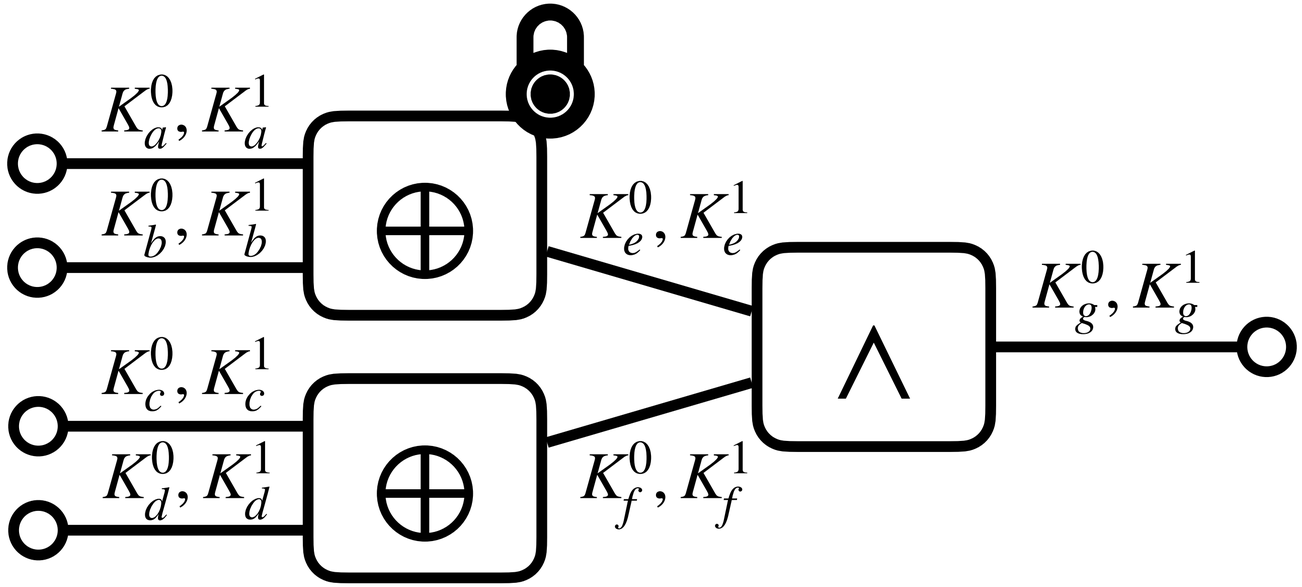


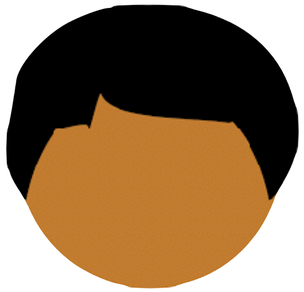
G



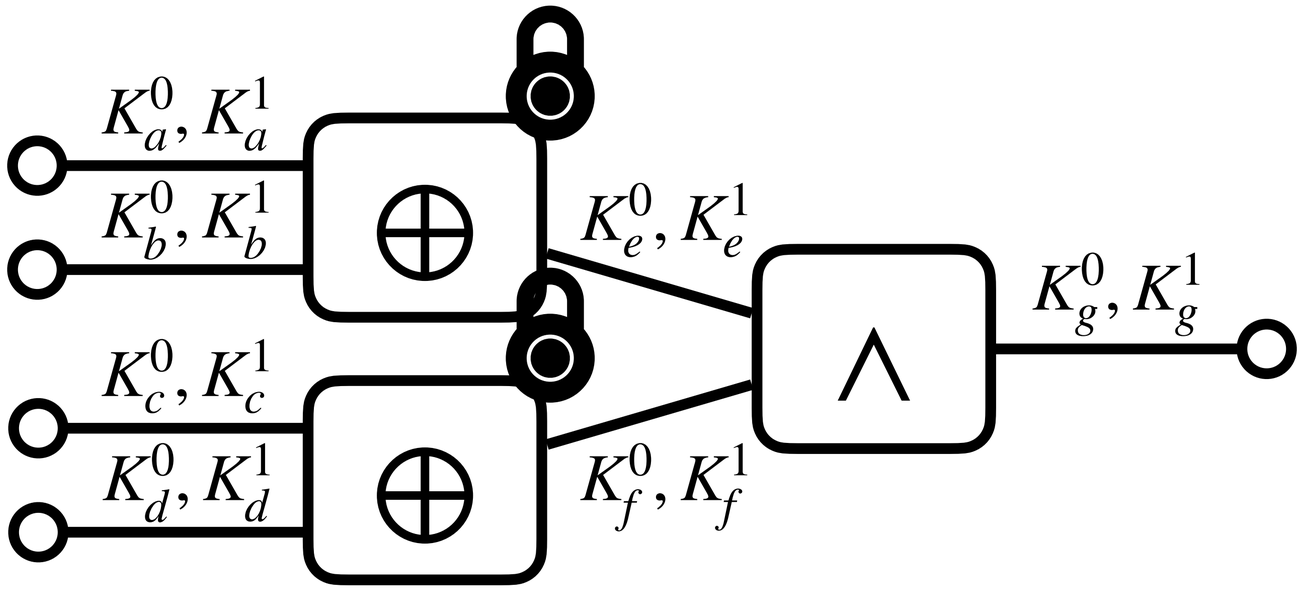


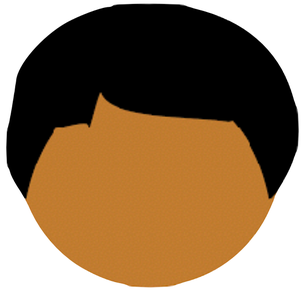
G



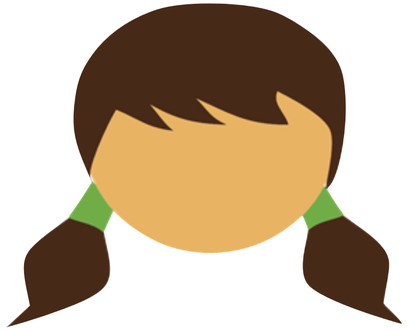


G

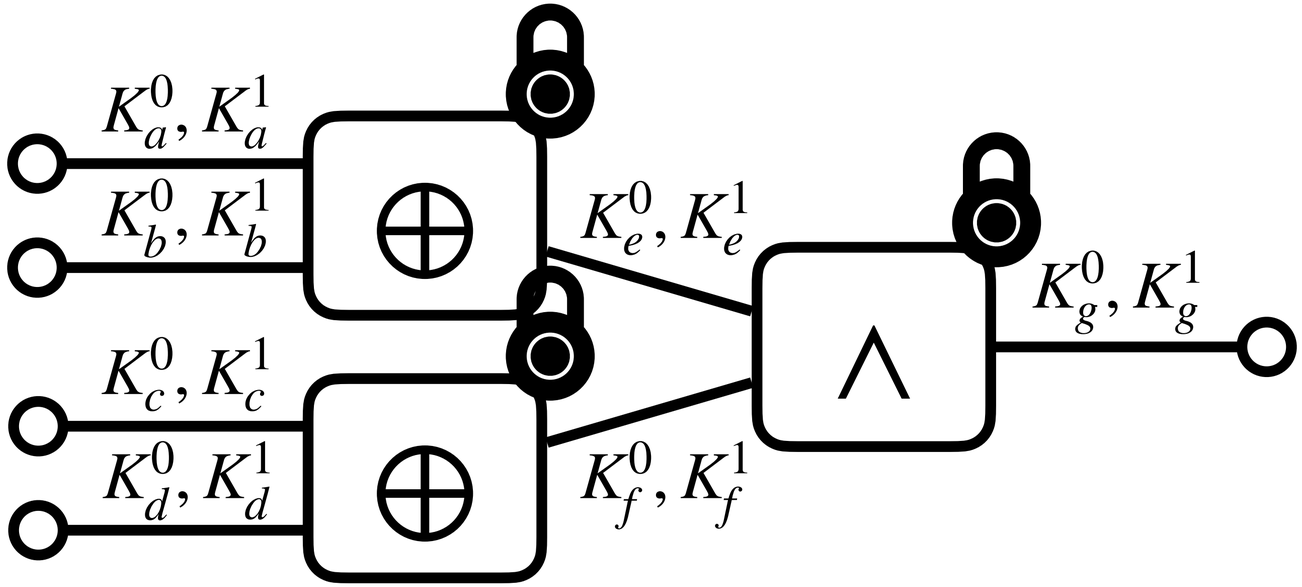


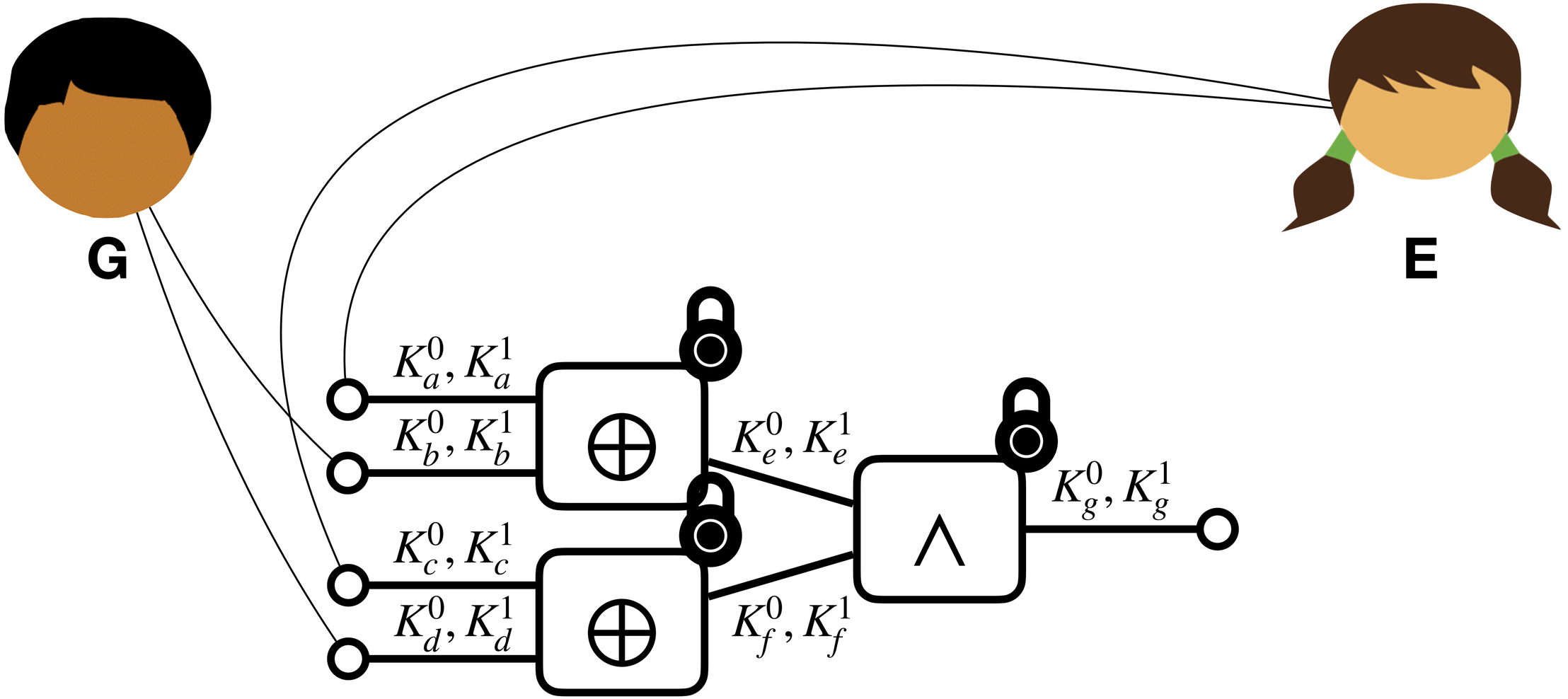


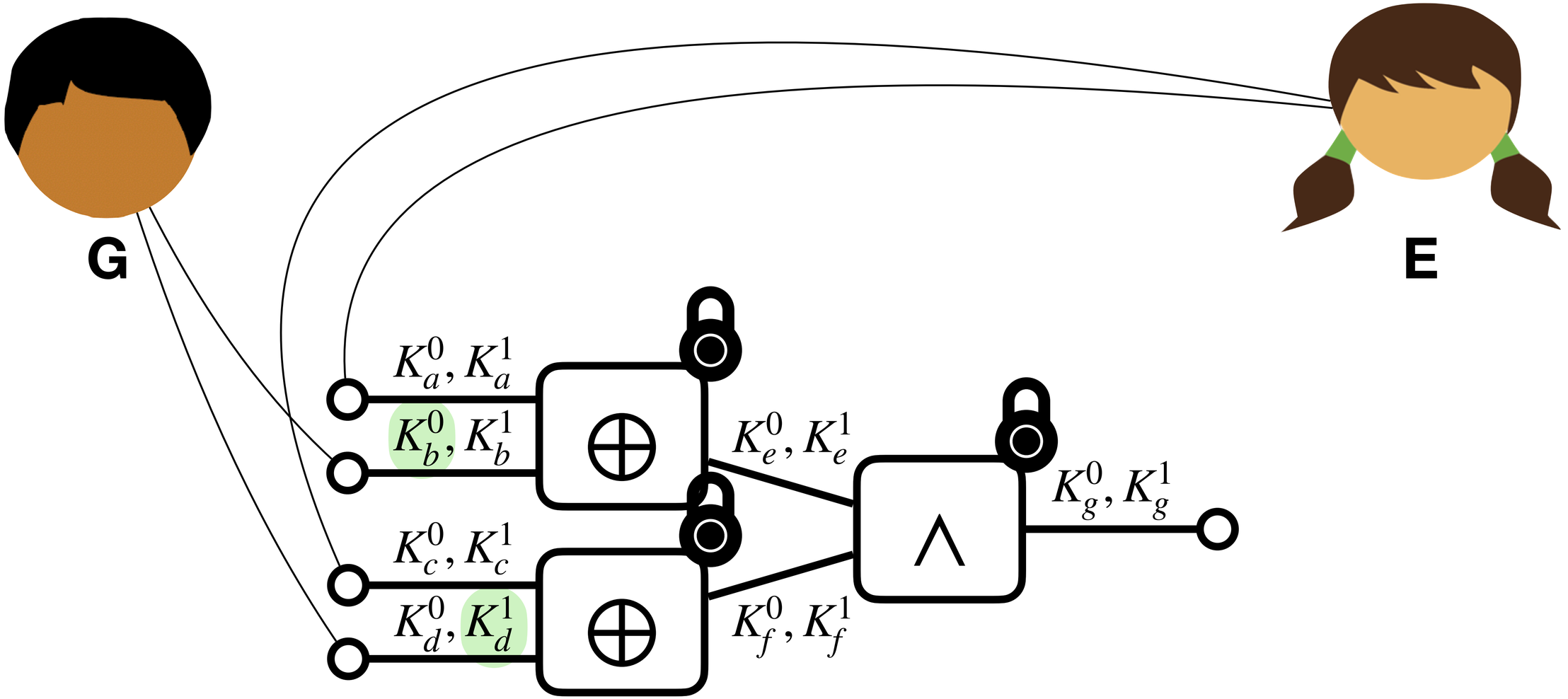
G

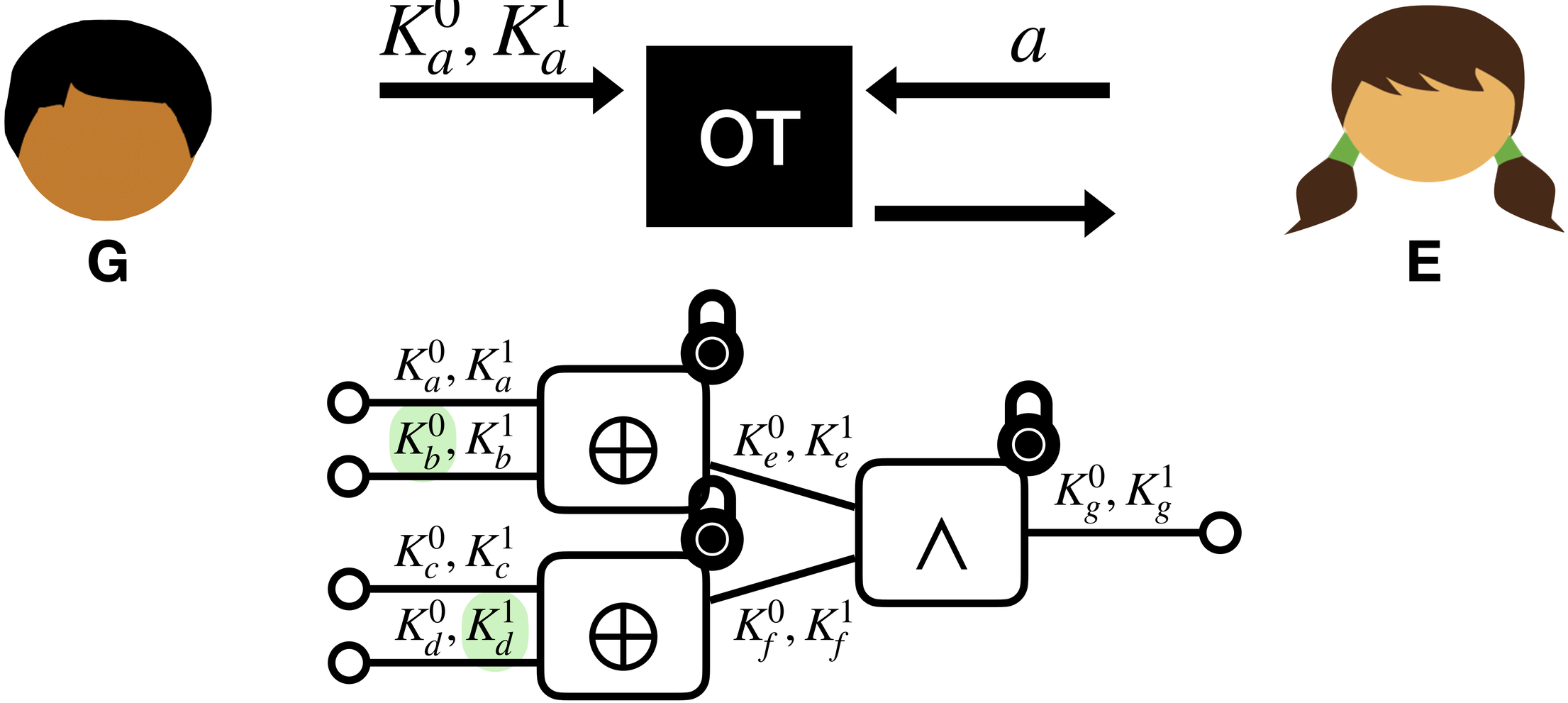


E

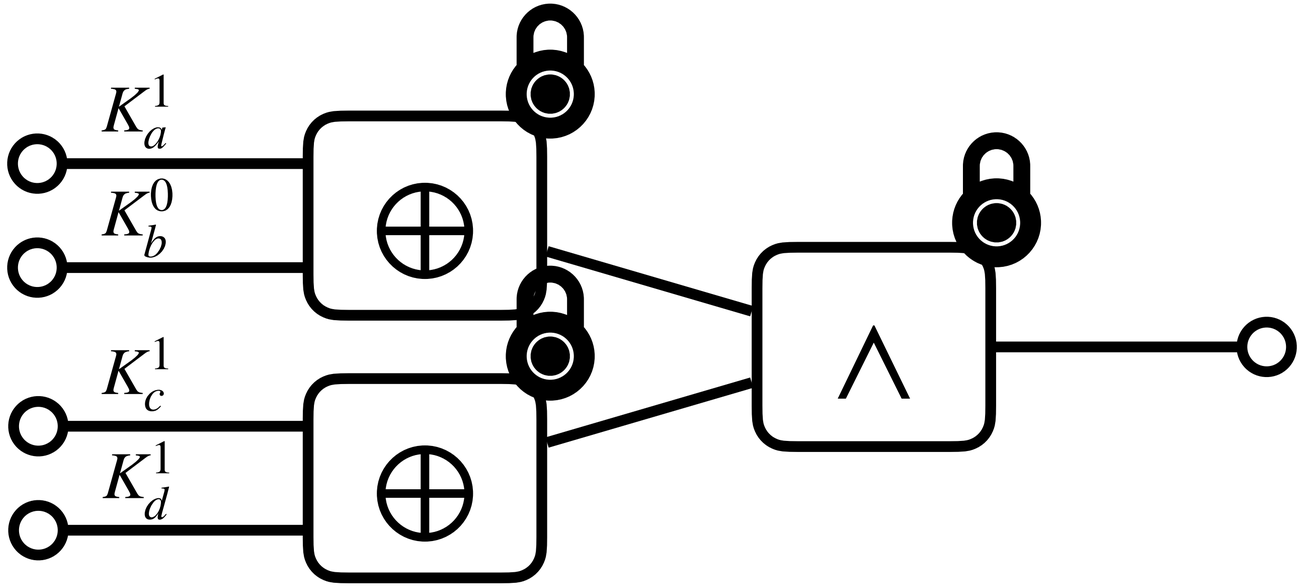

K_a^0, K_a^1
K_b^0, K_b^1
K_c^0, K_c^1
K_d^0, K_d^1
K_e^0, K_e^1
K_f^0, K_f^1
K_g^0, K_g^1
$\oplus$
$\oplus$
$\wedge$


G
E
K_a^0, K_a^1
K_b^0, K_b^1
K_c^0, K_c^1
K_d^0, K_d^1
K_e^0, K_e^1
K_f^0, K_f^1
K_g^0, K_g^1

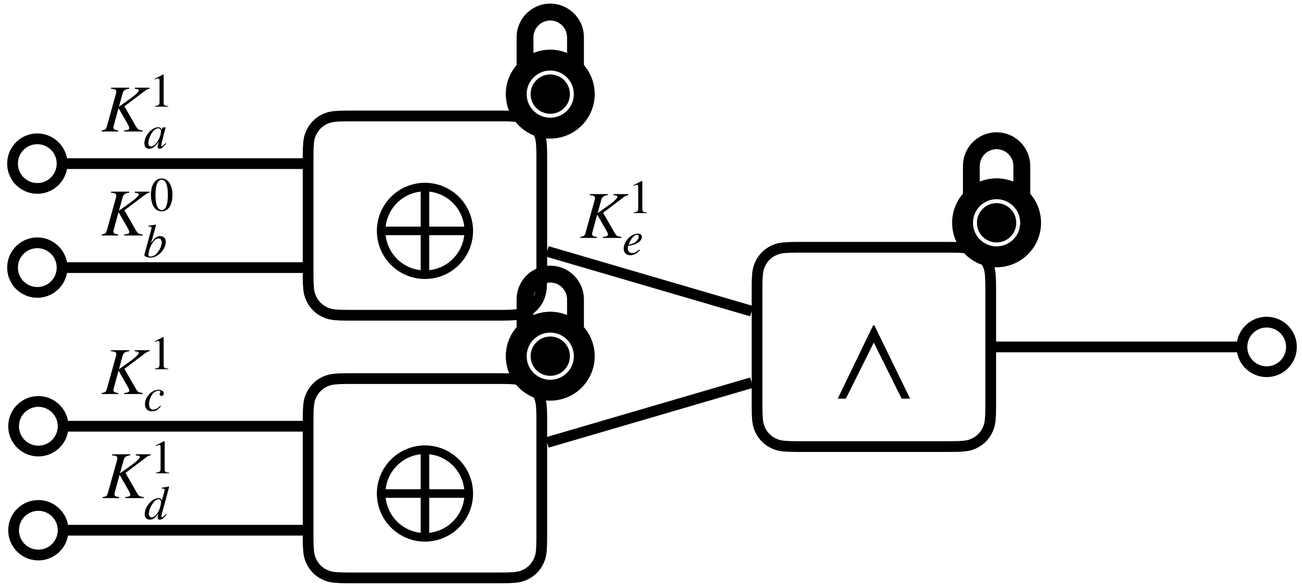

G
E
K_a^0, K_a^1
K_b^0, K_b^1
K_c^0, K_c^1
K_d^0, K_d^1
K_e^0, K_e^1
K_f^0, K_f^1
K_g^0, K_g^1

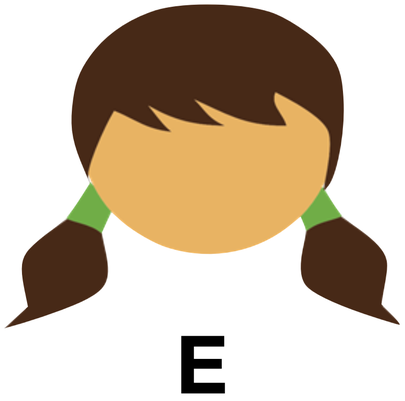

K_a^0, K_a^1
OT
a
G
E
K_a^0, K_a^1
K_b^0, K_b^1
K_c^0, K_c^1
K_d^0, K_d^1
K_e^0, K_e^1
K_f^0, K_f^1
K_g^0, K_g^1
$\oplus$
$\oplus$
$\wedge$

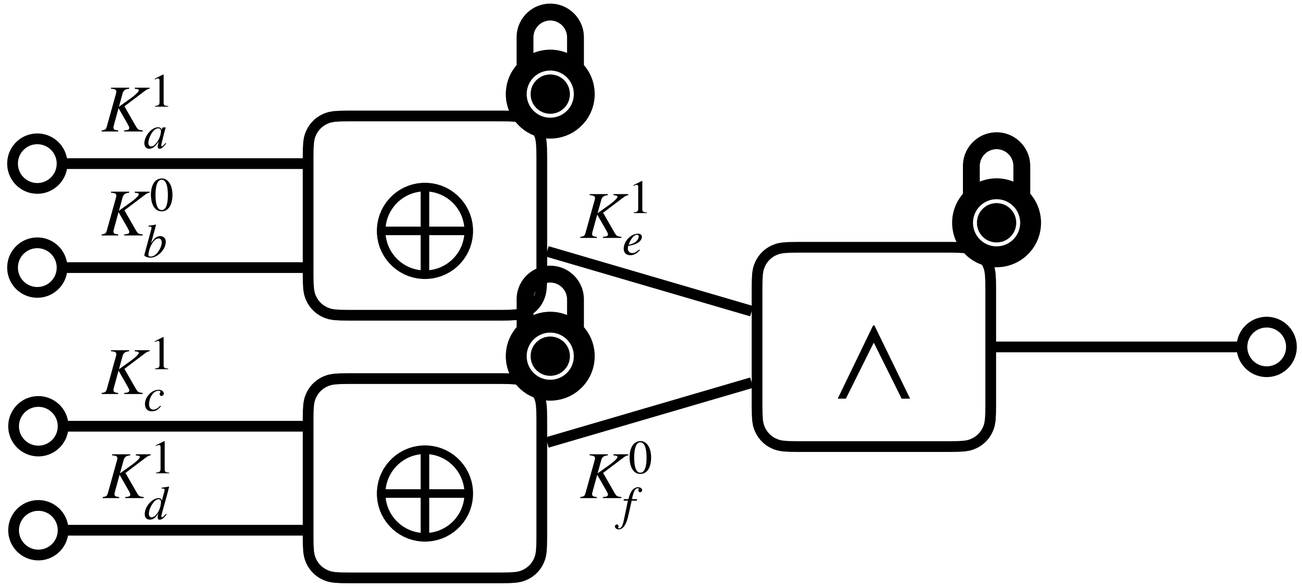

E

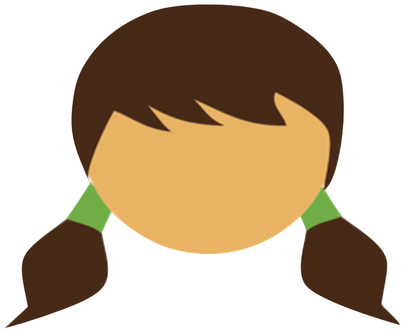

K_a^1
K_b^0
K_c^1
K_d^1
⊕
⊕
∧


E

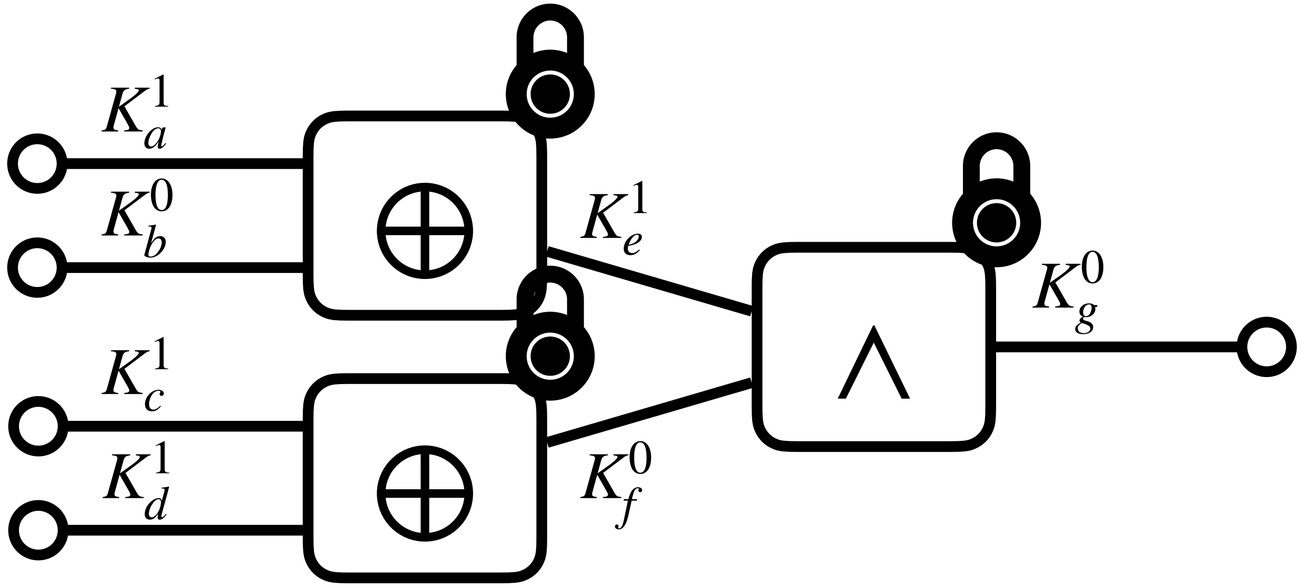

K_a^1
K_b^0
K_c^1
K_d^1
K_e^1
⊕
⊕
∧

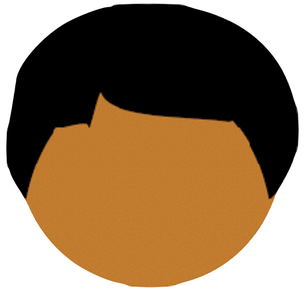

E


K_a^1
K_b^0
K_c^1
K_d^1
K_e^1
K_f^0



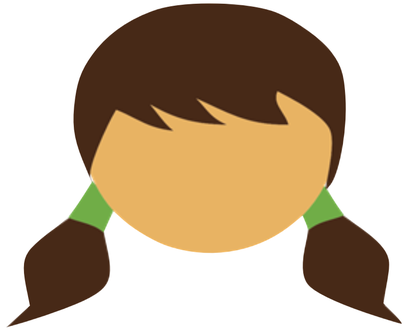
E



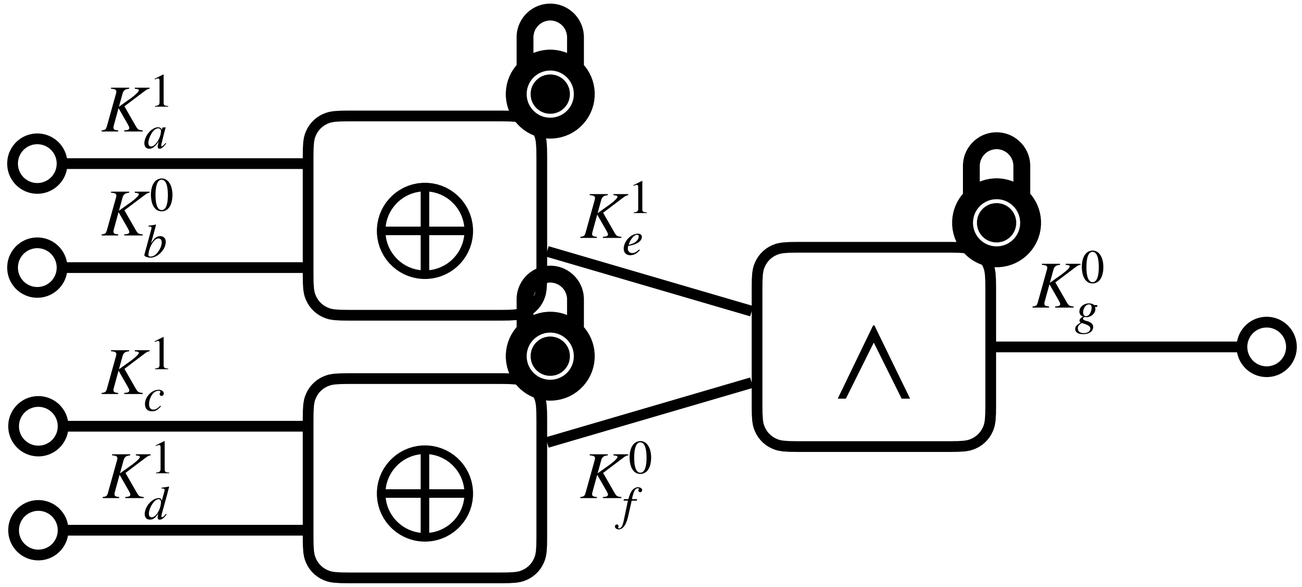


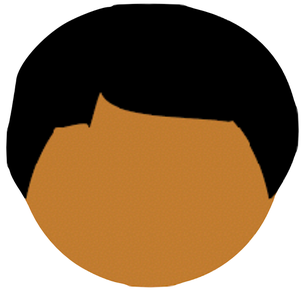
G

Encoding	Value
K_g^0	0
K_g^1	1



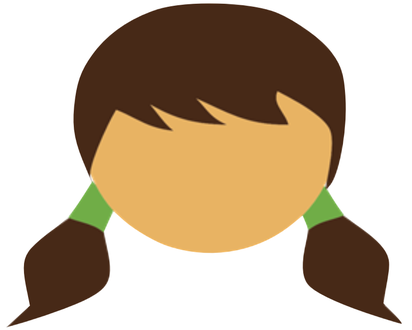
E



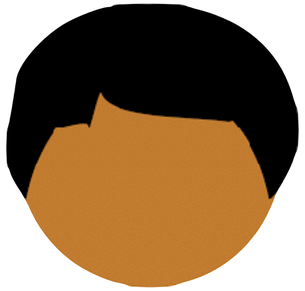


G

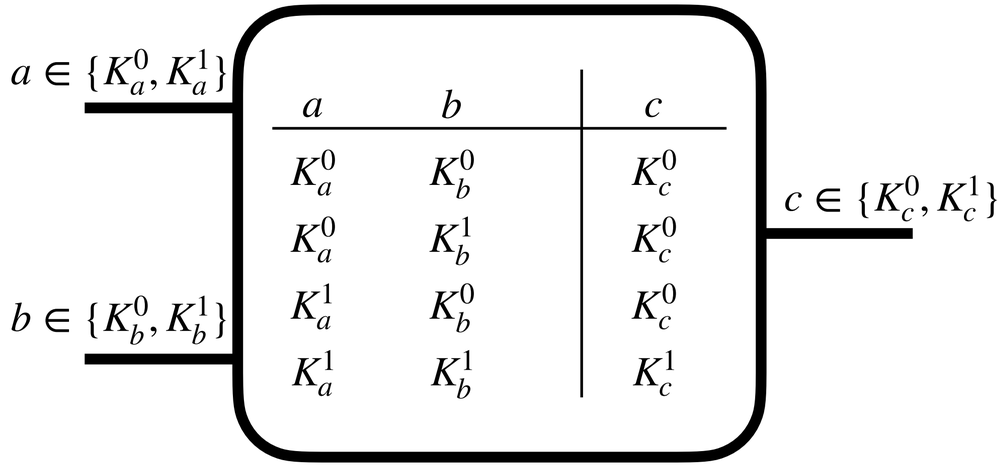
1. G “garbles” the circuit
2. G and E run OT so that E can select keys for her input
3. G sends keys for his input
4. G sends the garbled circuit and an output decoding table
5. E evaluates the circuit by decrypting rows
6. E learns the output and shares it with G



E



G

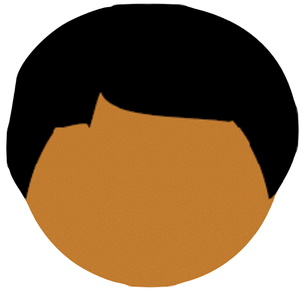


$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$

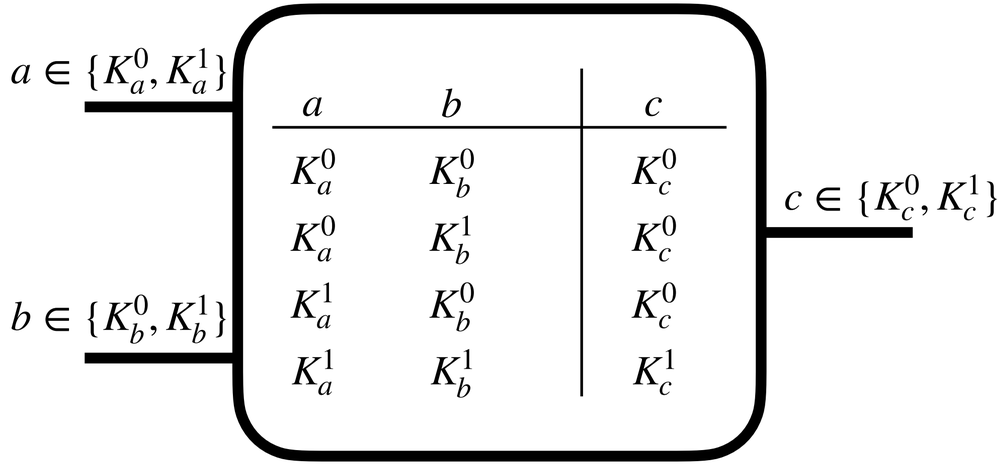
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$

$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$

$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

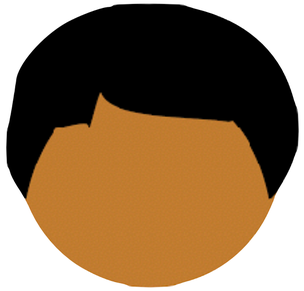


G

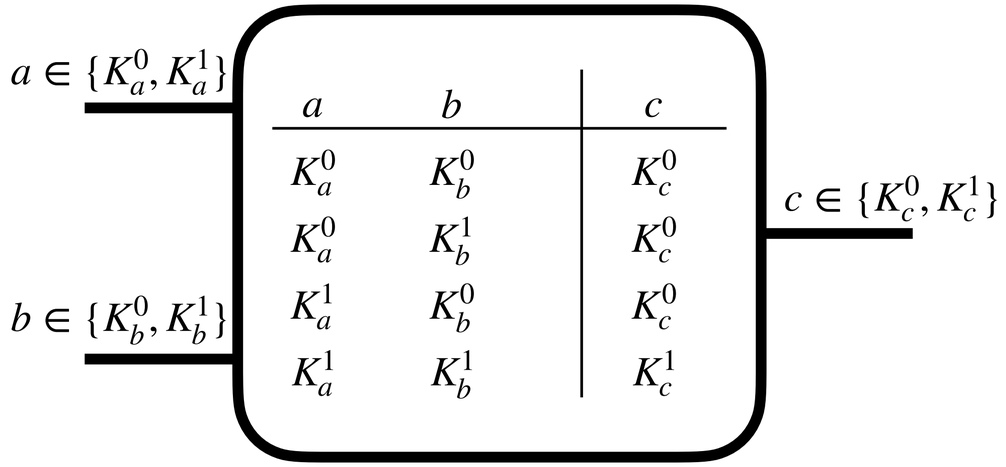


Problem:
If E knows which row to decrypt, this table reveals the value of input wires

$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$



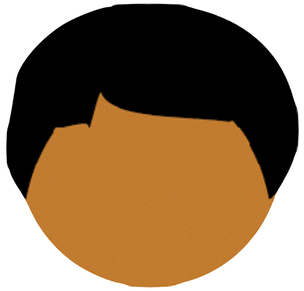
G



Problem:
If E knows which row to decrypt, this table reveals the value of input wires

$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

Solution:
G permutes rows before sending them



G

$a \in \{K_a^0, K_a^1\}$

$b \in \{K_b^0, K_b^1\}$

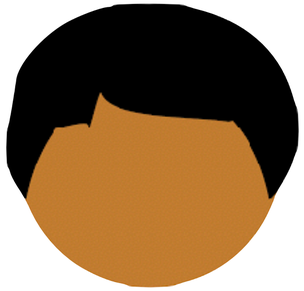
a	b	c
K_a^0	K_b^0	K_c^0
K_a^0	K_b^1	K_c^0
K_a^1	K_b^0	K_c^0
K_a^1	K_b^1	K_c^1

$c \in \{K_c^0, K_c^1\}$

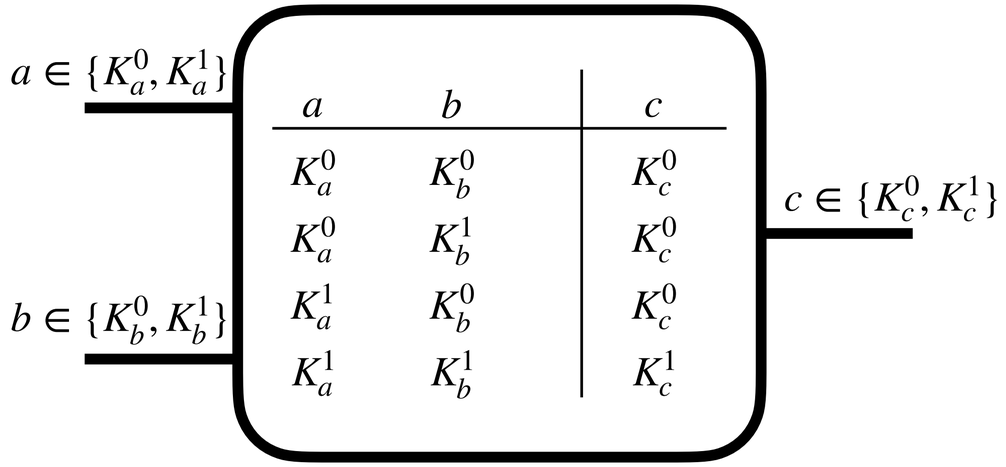
Problem:
If E knows which row to decrypt, this table reveals the value of input wires

$$\text{Enc}(K_a^1, \text{Enc}(K_b^0, K_c^0))$$
$$\text{Enc}(K_a^1, \text{Enc}(K_b^1, K_c^1))$$
$$\text{Enc}(K_a^0, \text{Enc}(K_b^0, K_c^0))$$
$$\text{Enc}(K_a^0, \text{Enc}(K_b^1, K_c^0))$$

Solution:
G permutes rows before sending them



G



Problem:
If E knows which row to decrypt, this table reveals the value of input wires

Solution:
G permutes rows before sending them

$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$
$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$

Problem:
How does E know which row to decrypt?

Solution:
Various solutions exist; e.g., add an additional pointer bit on each key

GMW

Garbled Circuit

GMW

Multi round

Garbled Circuit

Constant round

GMW

Multi round

Natural Extension to multiple parties

Garbled Circuit

Constant round

Natural only for 2PC

GMW

Multi round

Natural Extension to multiple parties

Low bandwidth with modern OT optimizations

Garbled Circuit

Constant round

Natural only for 2PC

High bandwidth

GMW

Multi round

Natural Extension to multiple parties

Low bandwidth with modern OT optimizations

Semi-honest

Garbled Circuit

Constant round

Natural only for 2PC

High bandwidth

Provides natural protection against malicious E

Security?

Two-Party Semi-Honest Security

Let f be a functionality. We say that a protocol Π securely computes f in the presence of a semi-honest adversary if for each party $i \in \{0,1\}$ there exists a polynomial time simulator $\mathcal{S}_i$ *such that for all inputs x_0, x_1:*

$$\{\text{View}_i^{\Pi}(x_0, x_1), \text{Output}^{\Pi}(x_0, x_1)\}$$

$$\overset{c}{=}$$

$$\{\mathcal{S}_i(x_i, y_i), (y_0, y_1) \mid (y_0, y_1) \leftarrow f(x_0, x_1)\}$$

Security?

Security against a semi-honest garbler is straightforward in the OT hybrid model

Security against evaluator is nuanced, since we must prove she learns nothing from the circuit encryption

A Proof of Security of Yao's Protocol for Two-Party Computation

Yehuda Lindell* Benny Pinkas†

June 26, 2006

Abstract

In the mid 1980's, Yao presented a constant-round protocol for securely computing any two-party functionality in the presence of semi-honest adversaries (FOCS 1986). In this paper, we provide a complete description of Yao's protocol, along with a rigorous proof of security. Despite the importance of Yao's protocol to the theory of cryptography, and in particular to the field of secure computation, to the best of our knowledge, this is the first time that an explicit proof of security has been published.

1 Introduction

In the setting of two-party computation, two parties with respective private inputs x and y, wish to jointly compute a functionality $f(x,y) = (f_1(x,y), f_2(x,y))$, such that the first party receives $f_1(x,y)$ and the second party receives $f_2(x,y)$. This functionality may be probabilistic, in which case $f(x,y)$ is a random variable. Loosely speaking, the security requirements are that nothing is learned from the protocol other than the output (*privacy*), and that the output is distributed according to the prescribed functionality (*correctness*). The definition of security that has become standard today [10, 11, 1, 4] blends these two conditions. In this paper, we consider the problem of achieving security in the presence of *semi-honest* (or passive) adversaries who follow the protocol specification, but attempt to learn additional information by analyzing the transcript of messages received during the execution.

The first general solution for the problem of secure two-party computation in the presence of semi-honest adversaries was presented by Yao [15]. Later, solutions were provided for the multi-party and malicious adversarial cases by Goldreich et al. [9]. These ground-breaking results essentially began the field of secure multiparty computation and served as the basis for countless papers. In addition to its fundamental theoretic contribution, Yao's protocol is remarkably efficient in that it has only a *constant number of rounds* and uses one oblivious transfer per input bit only (with no additional oblivious transfers in the rest of the computation). Unfortunately, to the best of our knowledge, a full proof of security of Yao's protocol has never been published. Our motivation for publishing such a proof is twofold. First, Yao's result is central to the field of secure computation. This is true both because of its historic importance as the first general solution to the two-party problem, and because many later results have relied on it in their constructions. As such, having a rigorous proof of the result is paramount. Second, the current situation is very frustrating for those who wish to study secure multiparty computation, but are unable to find a complete presentation of one of the most basic results in the field. We hope to correct this situation in this paper.

*Department of Computer Science, Bar-Ilan University, Israel. email: lindell@cs.biu.ac.il. Most of this work was carried out while at IBM T.J.Watson Research, New York.

†Department of Computer Science, Haifa University, Israel. email: benny@pinkas.net. Most of this work was carried out while at HP Labs, New Jersey.

Pseudorandom Function (PRF)

A function family F is considered pseudorandom *if the following indistinguishability holds*

Real:

$k \xleftarrow{\$} \{0,1\}^{\lambda}$

`lookup(`m`):`

`return` $F(k, m)$

$\overset{c}{=}$

Ideal:

$T \leftarrow \text{EmptyMap}$

`lookup(`m`):`

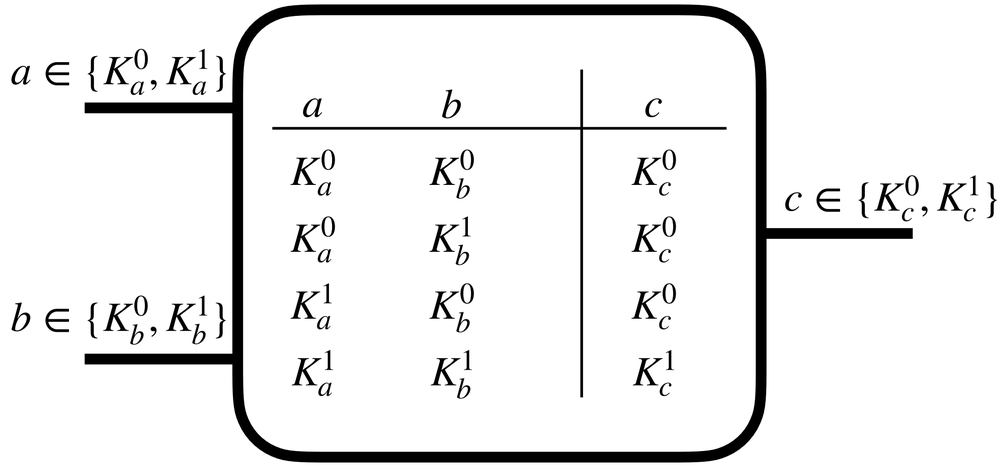
`if` $m \notin T$`:`

$T[m] \xleftarrow{\$} \{0,1\}^{\text{out}}$

`return` $T[m]$

“If you don’t know the key, F looks random”

E



a	b	c
K_a^0	K_b^0	K_c^0
K_a^0	K_b^1	K_c^0
K_a^1	K_b^0	K_c^0
K_a^1	K_b^1	K_c^1

$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$$

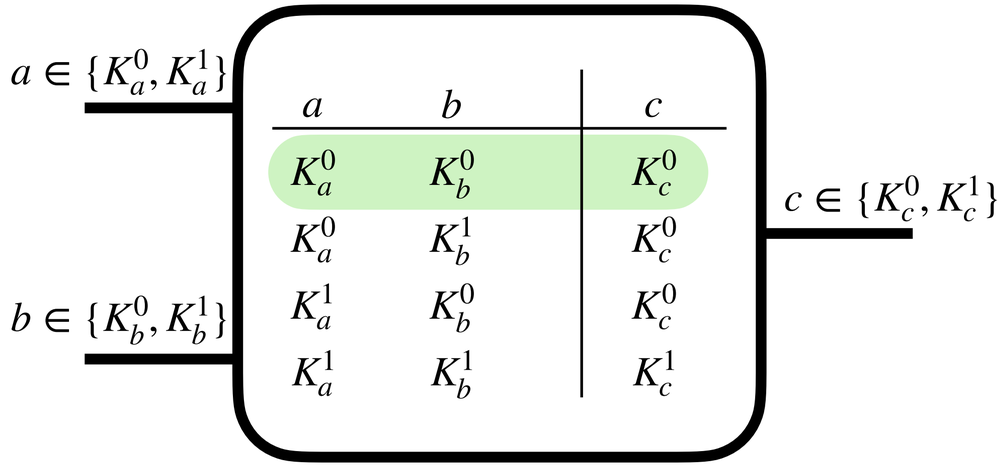
$$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$$

$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$$

$$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$$

E

K_a^0, K_b^0



$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^0, K_c^0))$

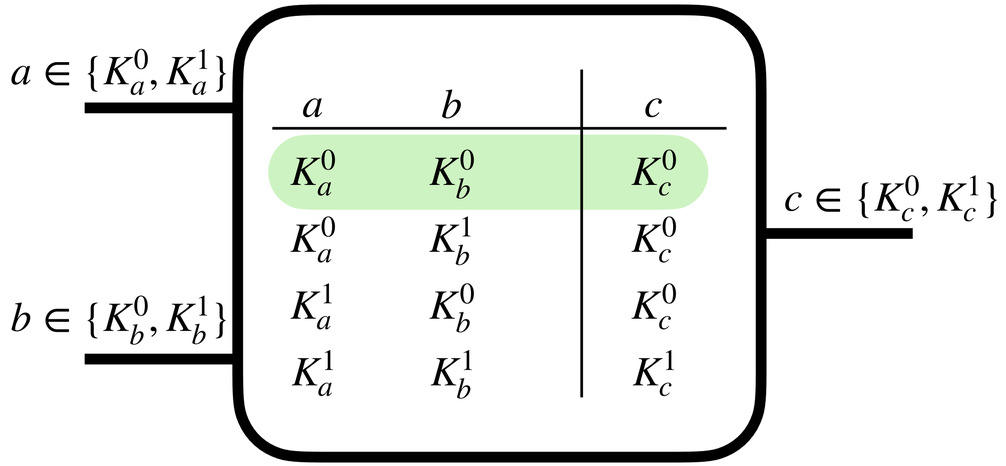
$\mathrm{Enc}(K_a^1, \mathrm{Enc}(K_b^1, K_c^1))$

$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^0, K_c^0))$

$\mathrm{Enc}(K_a^0, \mathrm{Enc}(K_b^1, K_c^0))$

E

K_a^0, K_b^0



$F(K_a^1, F(K_b^0, K_c^0))$

$F(K_a^1, F(K_b^1, K_c^1))$

$F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, F(K_b^1, K_c^0))$

Use PRF to instantiate encryption

E

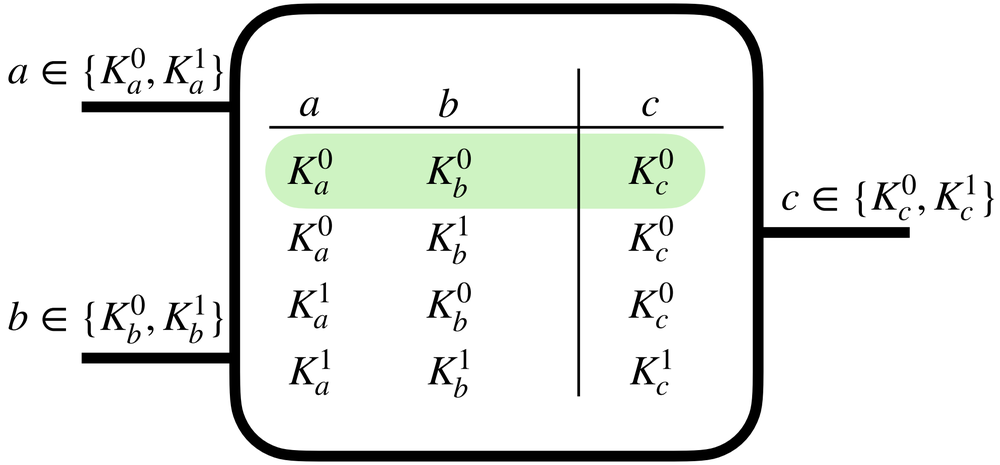
K_a^0, K_b^0

$F(K_a^1, F(K_b^0, K_c^0))$

$F(K_a^1, F(K_b^1, K_c^1))$

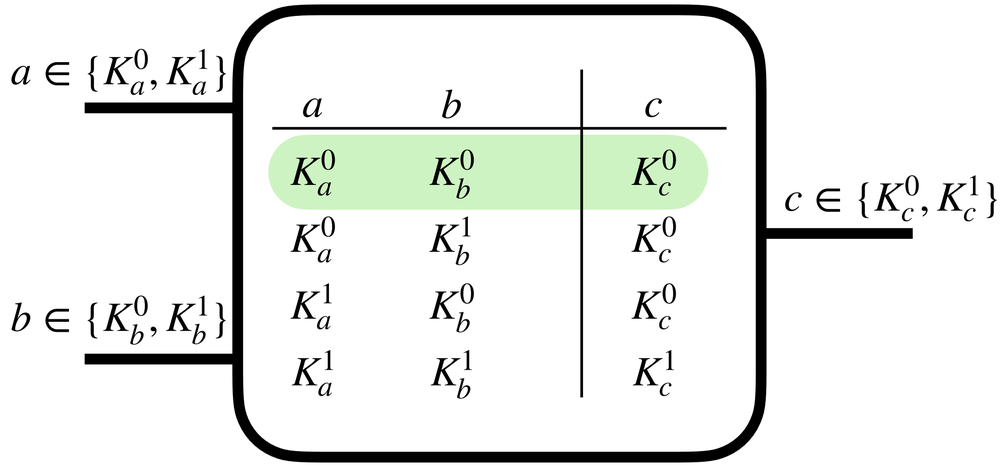
$F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, F(K_b^1, K_c^0))$



a	b	c
K_a^0	K_b^0	K_c^0
K_a^0	K_b^1	K_c^0
K_a^1	K_b^0	K_c^0
K_a^1	K_b^1	K_c^1

E



K_a^0, K_b^0

$F(K_a^1, F(K_b^0, K_c^0))$ random

$F(K_a^1, F(K_b^1, K_c^1))$ random

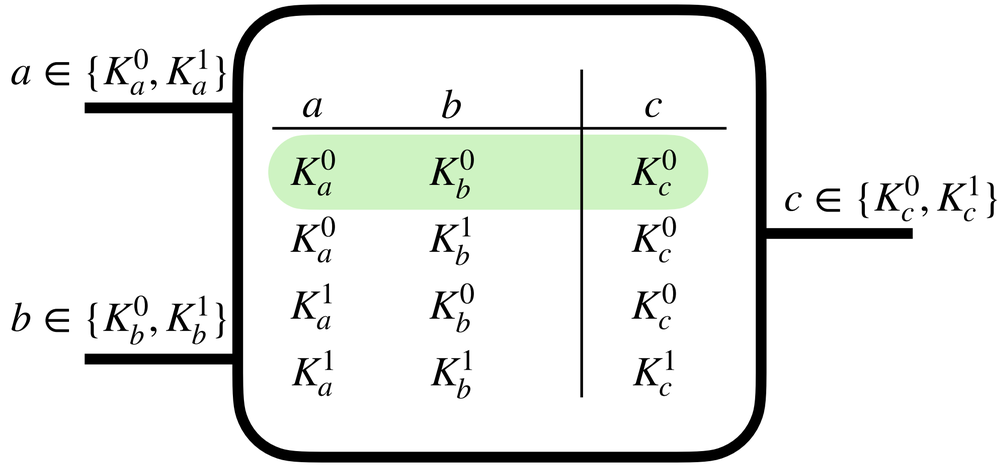
$F(K_a^0, F(K_b^0, K_c^0))$ $\overset{c}{=}$ $F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, F(K_b^1, K_c^0))$ $F(K_a^0, F(K_b^1, K_c^0))$

By PRF security; K_a^1 is not in E's view

E

K_a^0, K_b^0



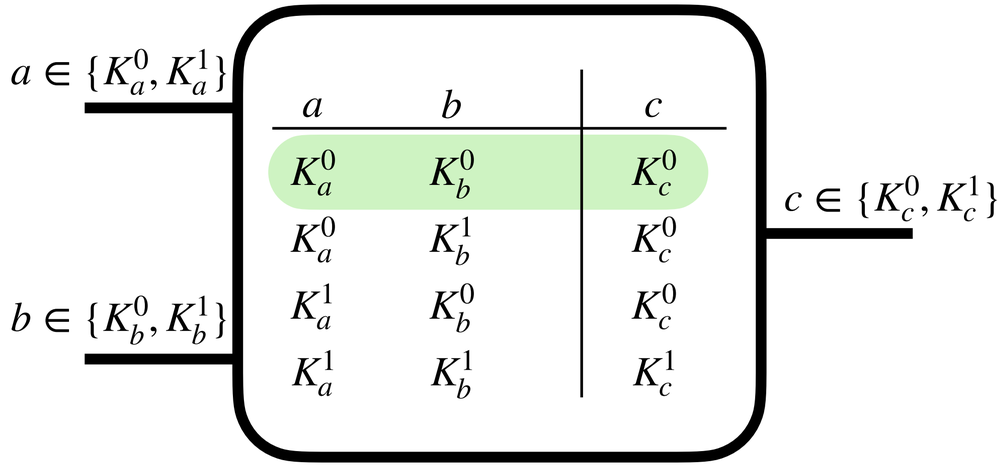
random

random

$F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, F(K_b^1, K_c^0))$

E



K_a^0, K_b^0

random

random

$F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, \text{random})$

$\overset{c}{=}$

random

random

$F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, F(K_b^1, K_c^0))$

By PRF security; K_b^1 is not in E's view

E

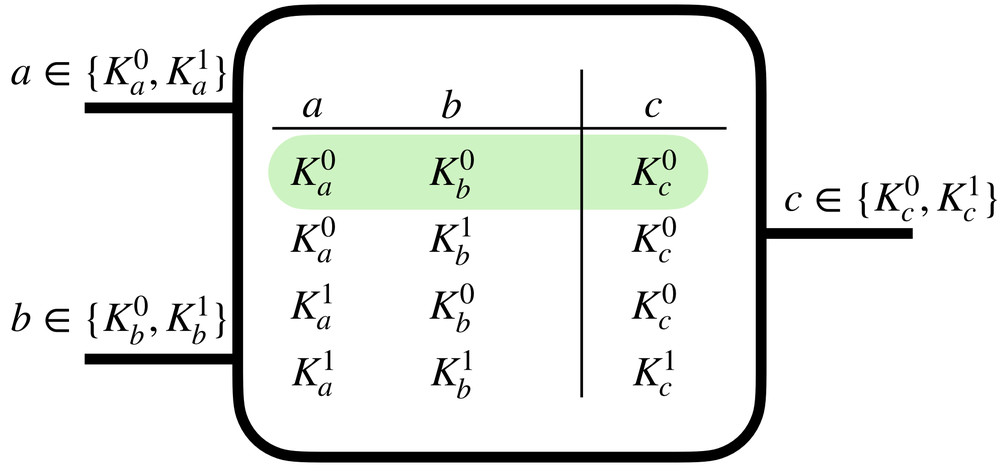
K_a^0, K_b^0

random

random

$F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, \text{random})$



Notice, mentions of K_c^1 are removed from the gate encryption!

E

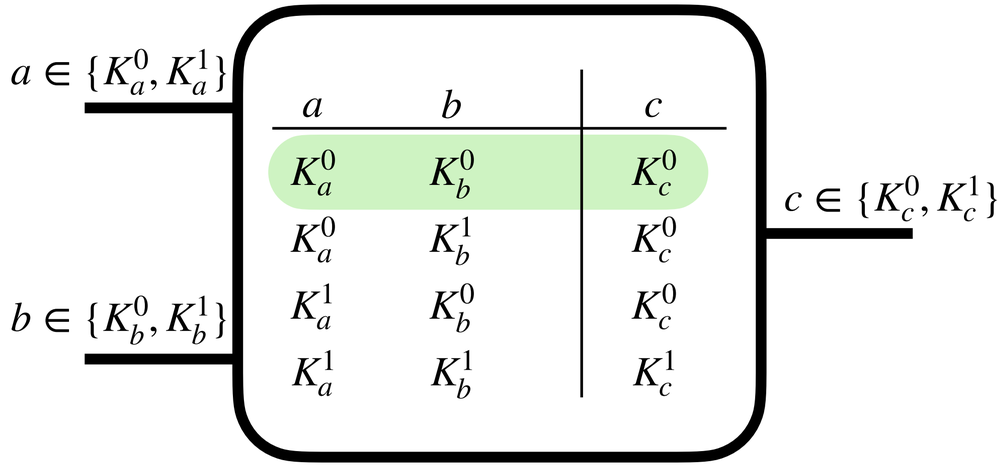
K_a^0, K_b^0

random

random

$F(K_a^0, F(K_b^0, K_c^0))$

$F(K_a^0, \text{random})$



Once all gates that take a, b as input are gone, K_a^0, K_b^0 are just uniform strings

E

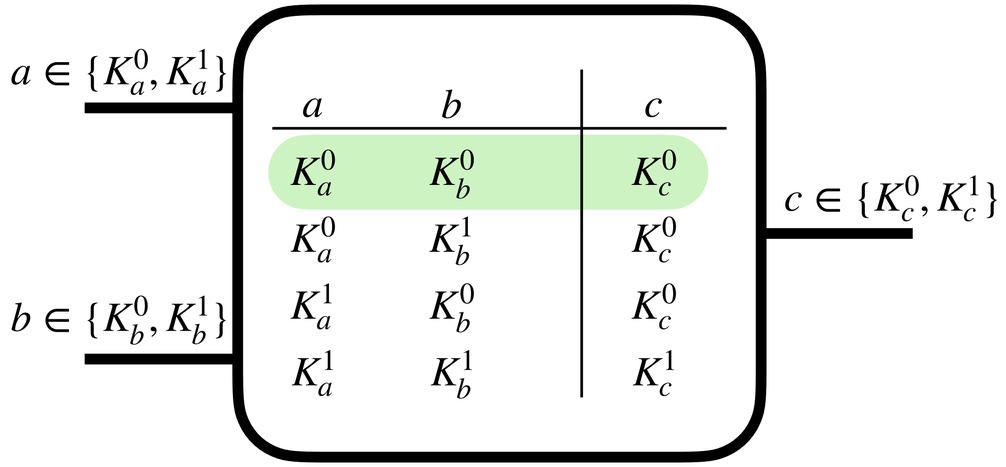
K_a, K_b

random

random

$F(K_a, F(K_b, K_c))$

$F(K_a, \text{random})$



Once all gates that take a, b as input are gone, K_a^0, K_b^0 are just uniform strings

E

K_a, K_b

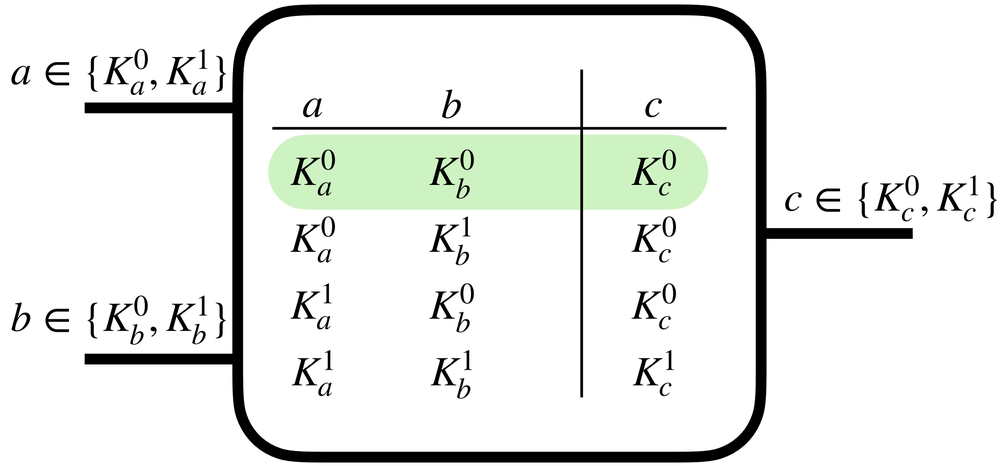
random

random

$F(K_a, F(K_b, K_c))$

$F(K_a, \text{random})$

Simulator outputs encrypted truth tables that look like this



Once all gates that take a, b as input are gone, K_a^0, K_b^0 are just uniform strings

Today’s objectives

Review GMW and its round complexity

Introduce Garbled Circuits

Discuss trade-offs between GMW and GC

Explore GC security proof