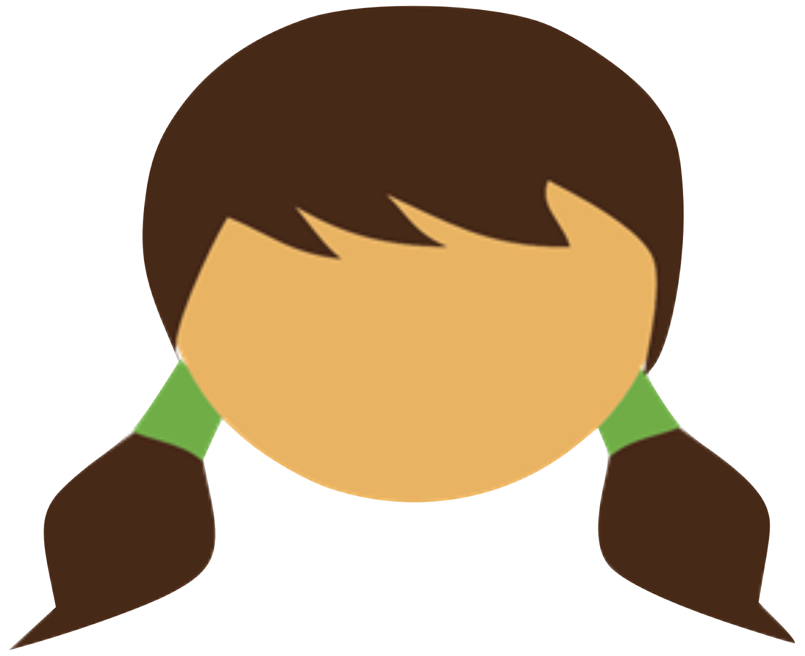
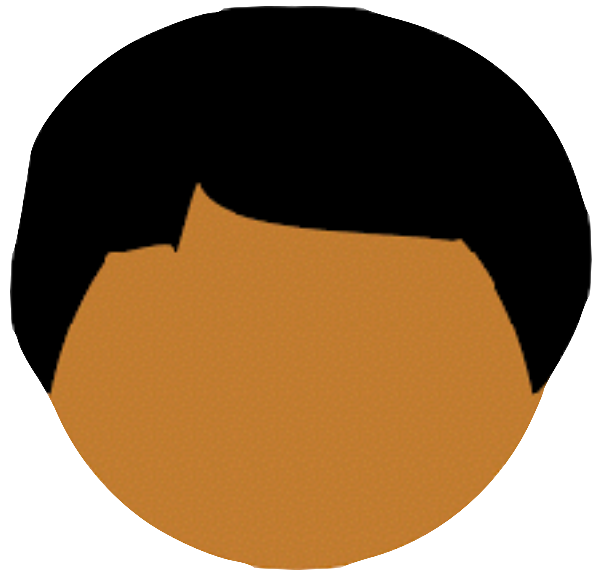
A Quick Tour of Secure Multiparty Computation (MPC)

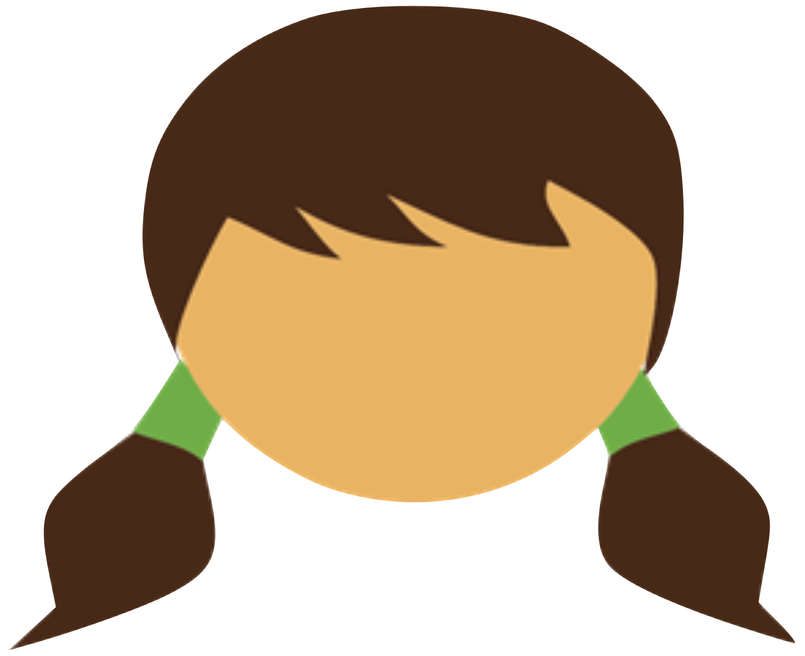
David Heath



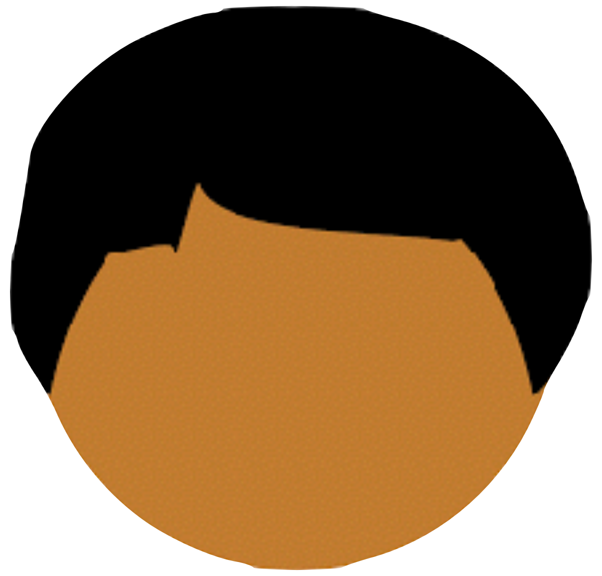


```
// F.c

int main (int argc,
  char** argv) {
    ...
}
```



x



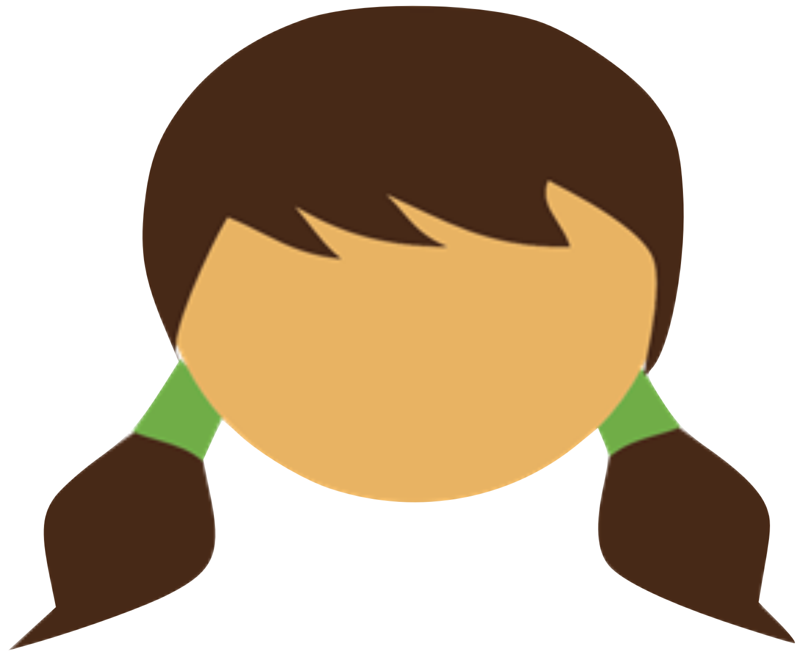
y

z

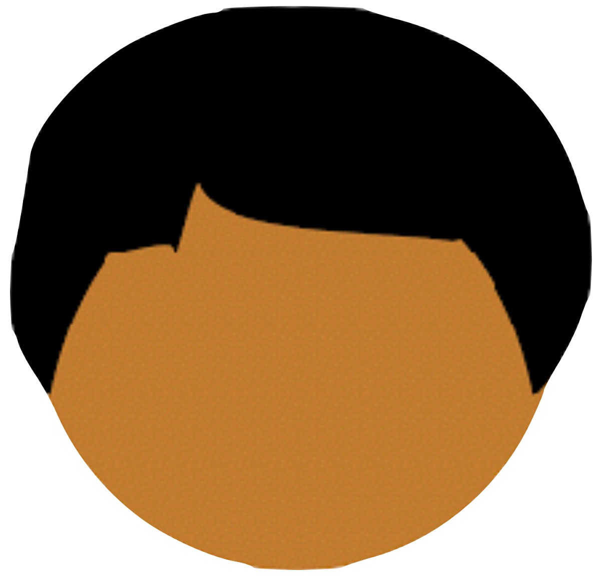
Trusted Third Party

```
// F.c

int main (int argc,
  char** argv) {
    ...
}
```



x



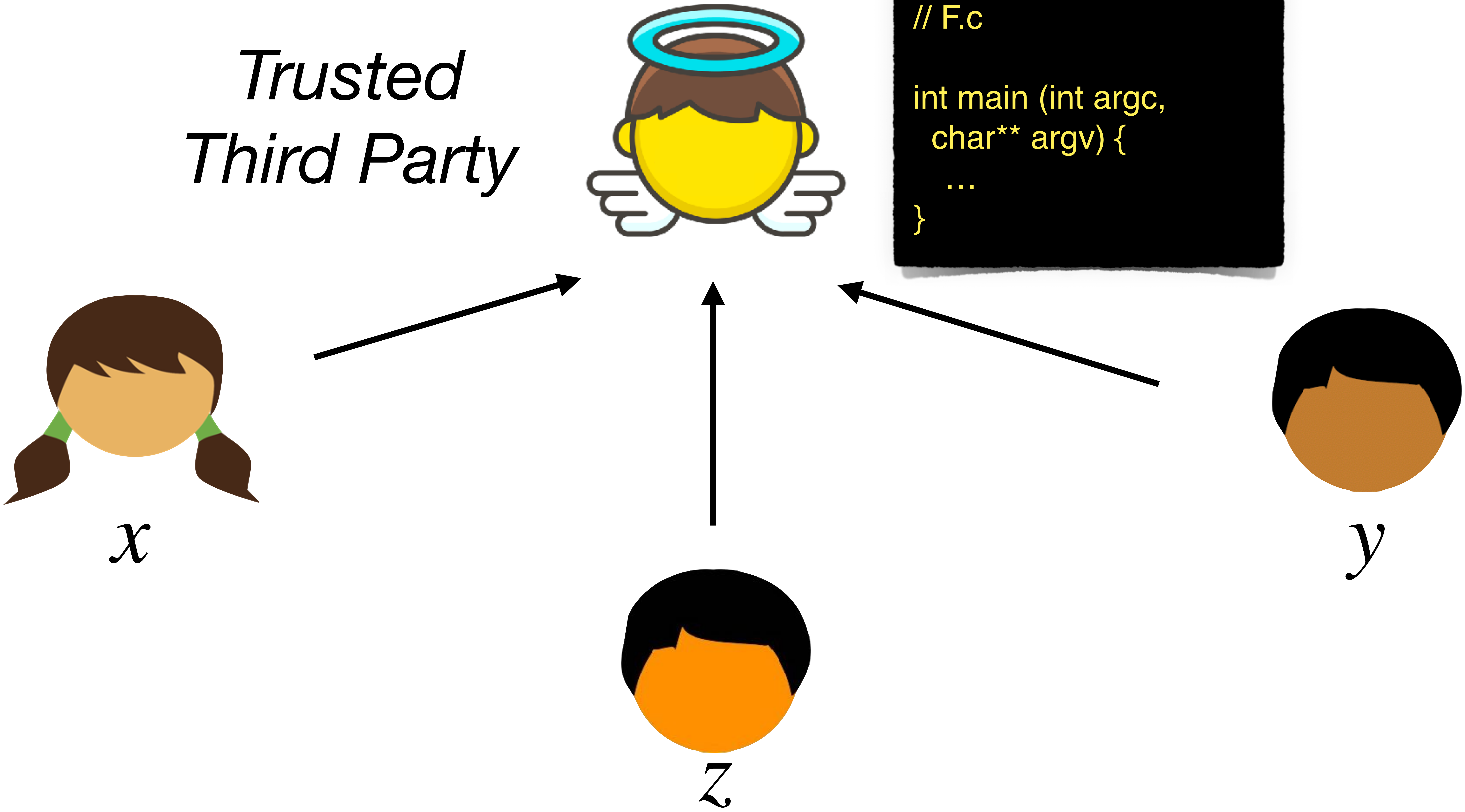
y

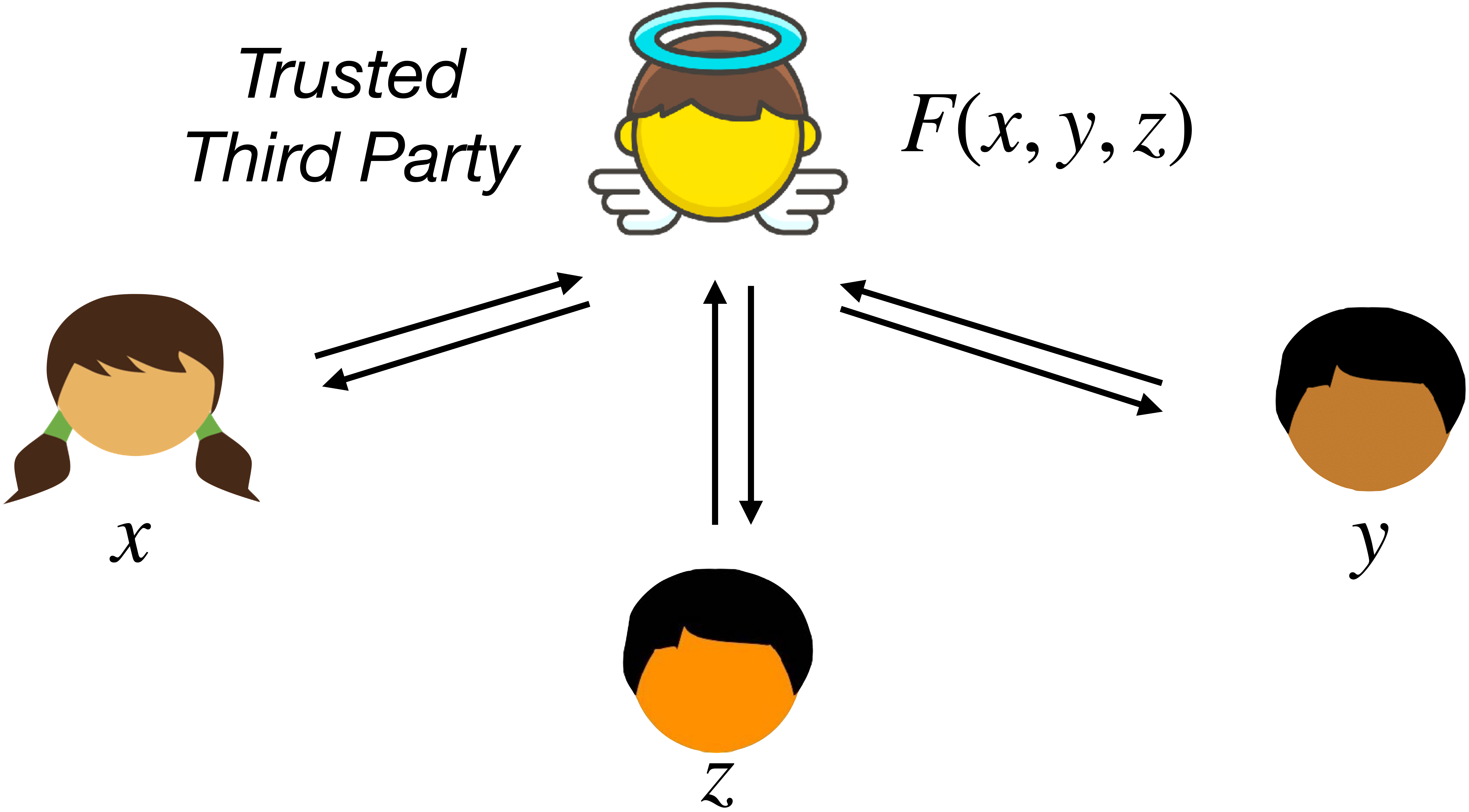
z

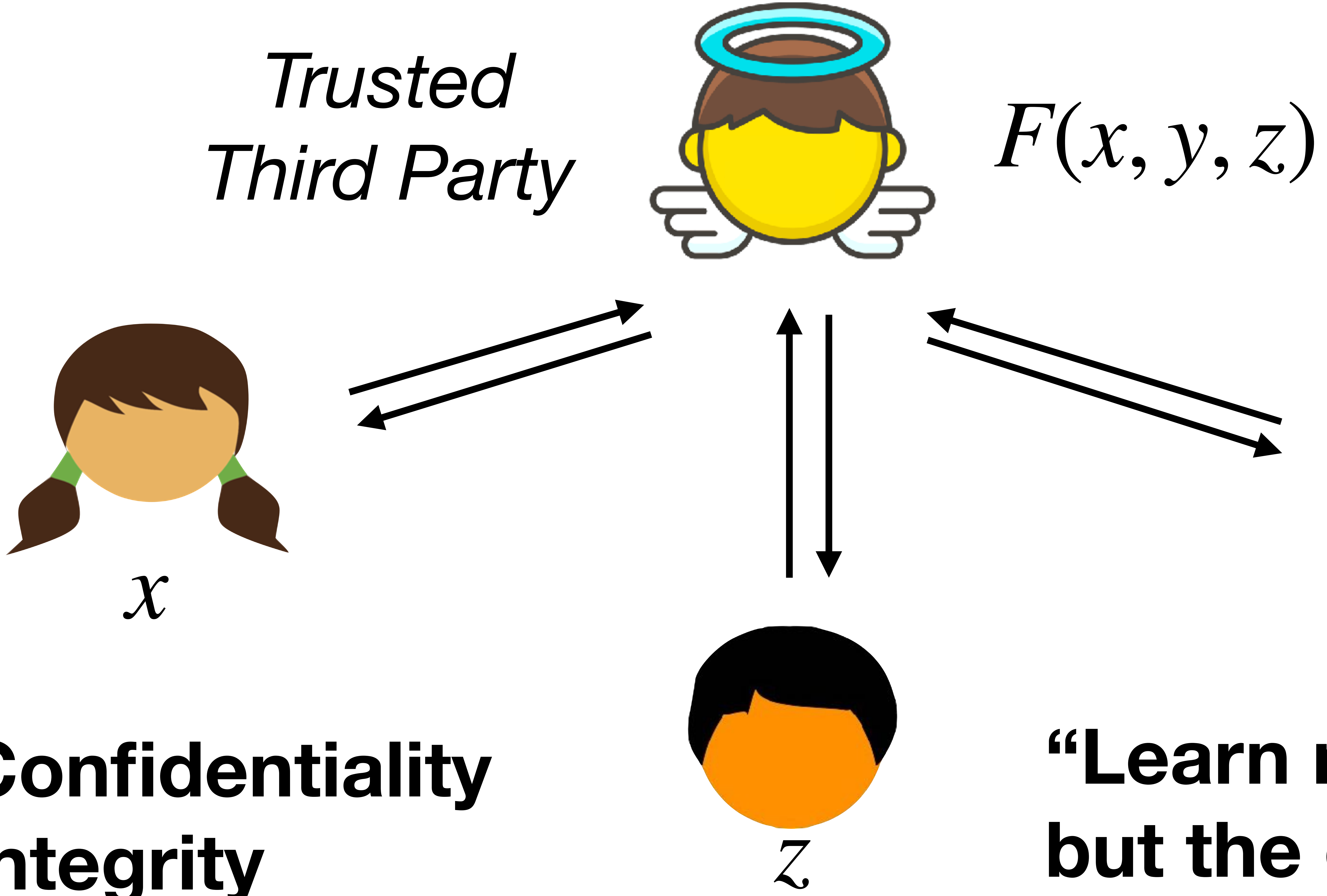
Trusted Third Party

```c
// F.c

int main (int argc,
  char** argv) {
    ...
}
```

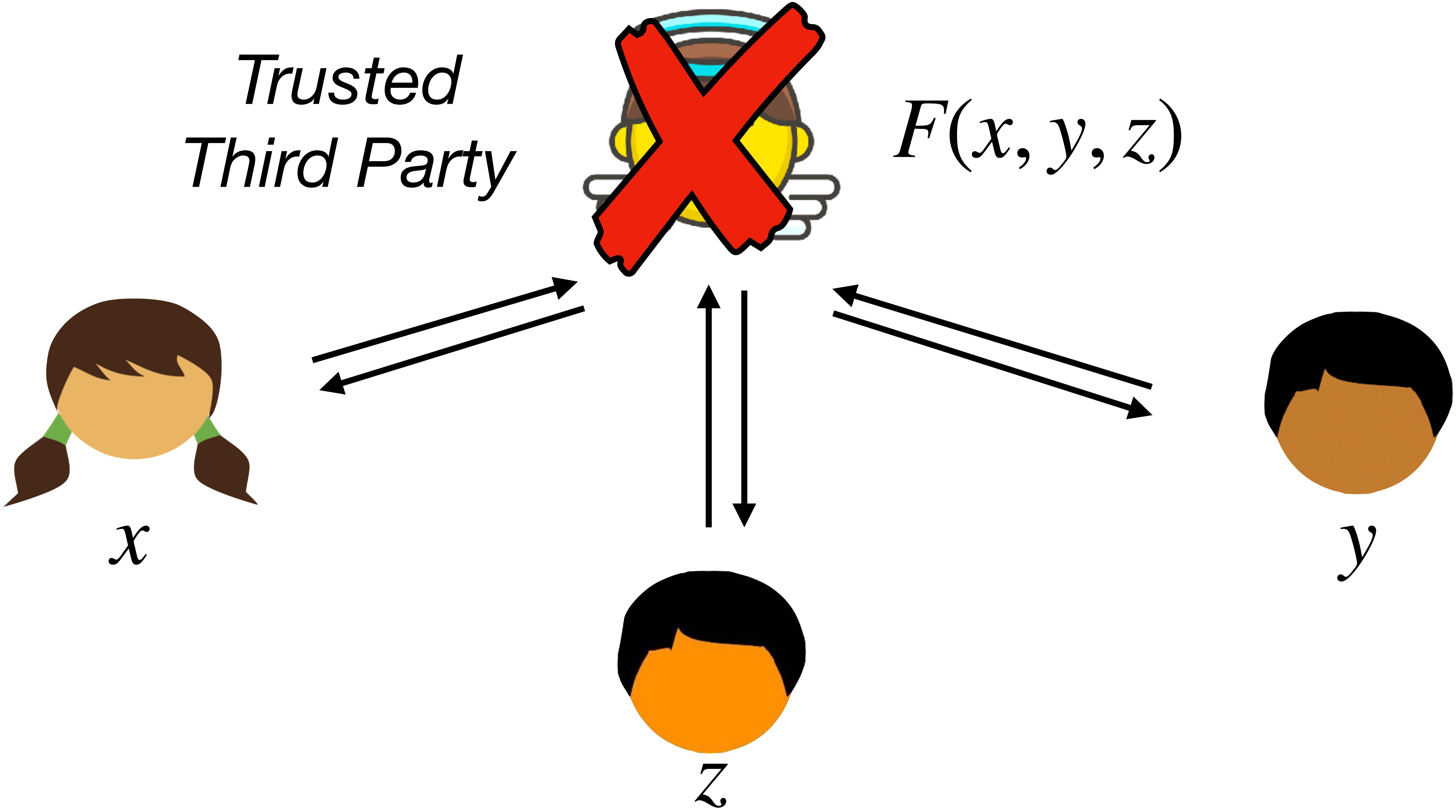


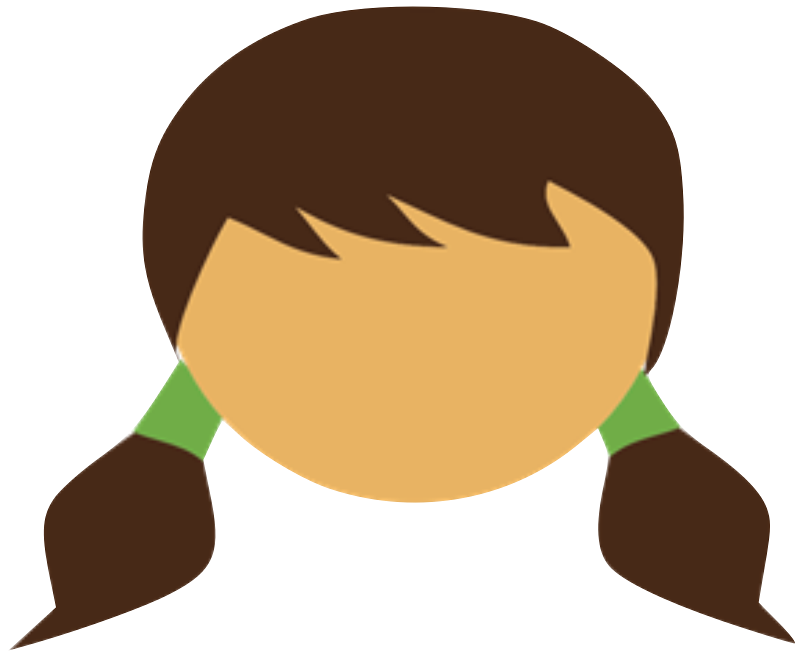

Trusted
Third Party
$F(x, y, z)$
x
y
z



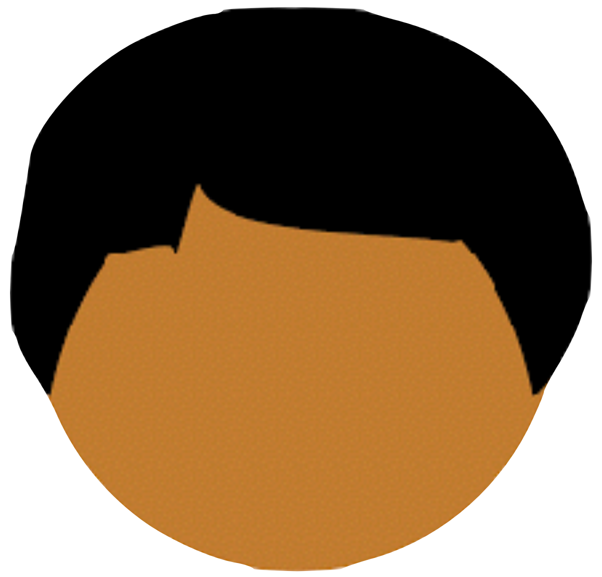
Confidentiality
Integrity

“Learn nothing but the output”


Trusted
Third Party
$F(x, y, z)$
x
y
z

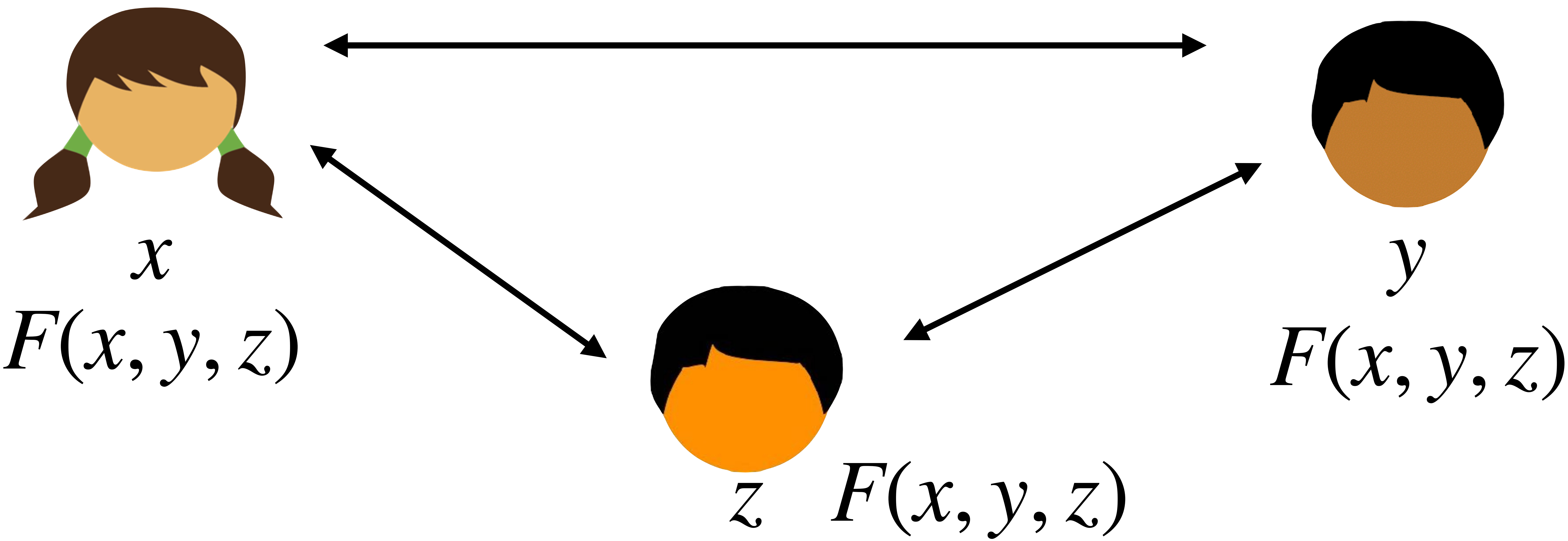


x



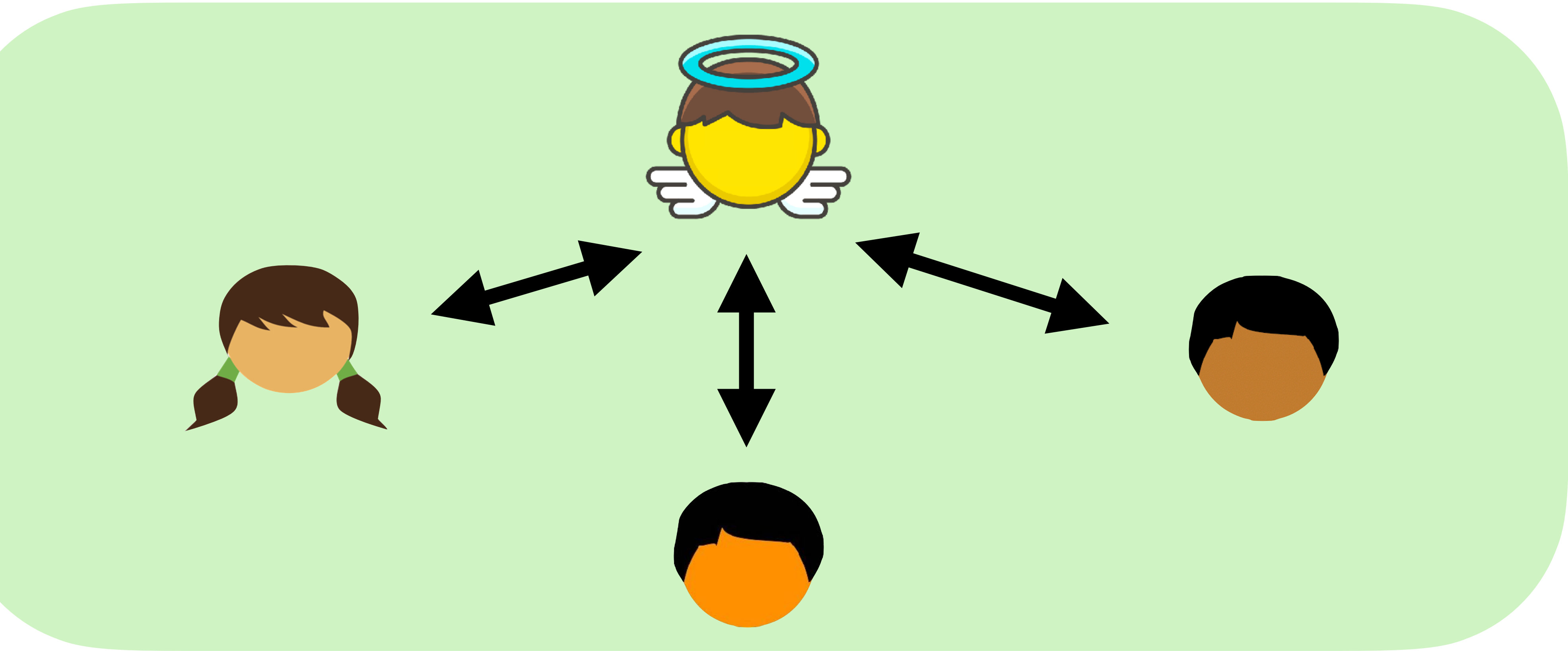
y

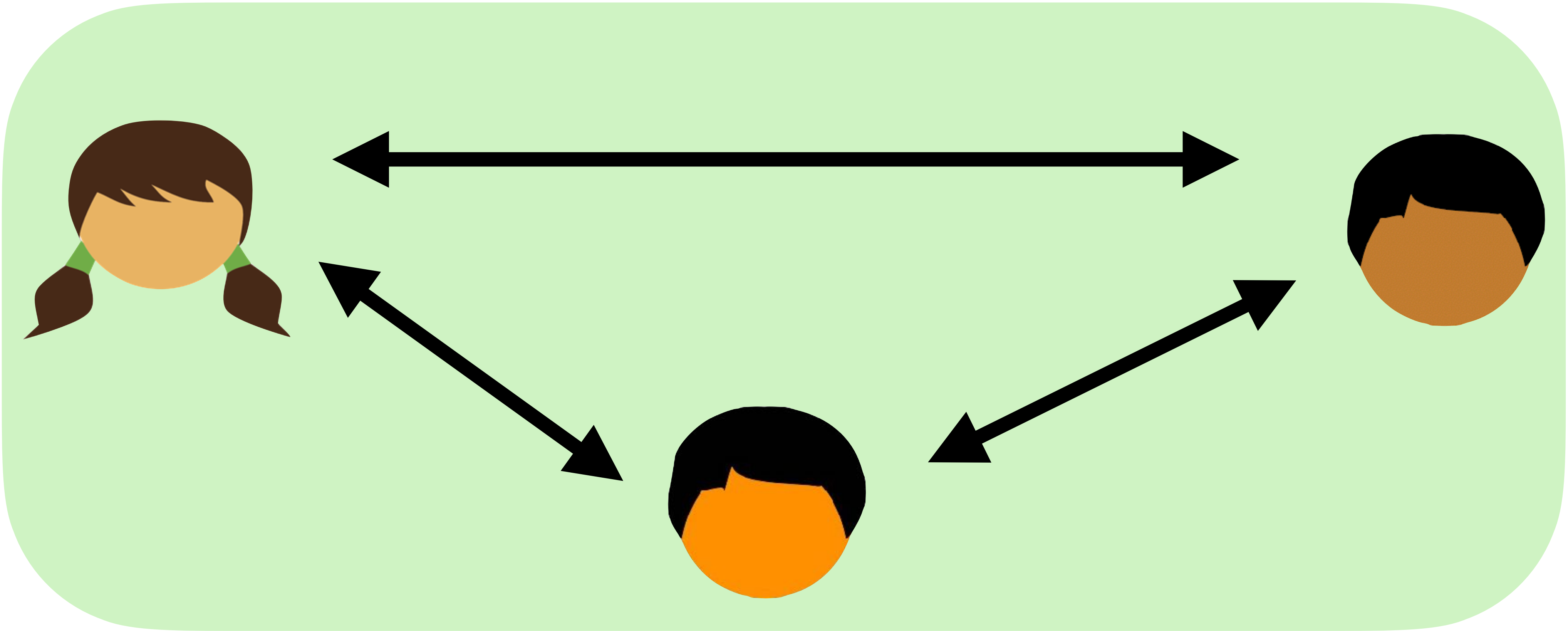
z



Confidentiality

Integrity





Secure Multiparty Computation

How can we use cryptography to ***emulate the existence of a trusted third party*** *so that we can run* ***arbitrary programs on joint private inputs?***

Secure Auctions

Secure Multiparty Computation Goes Live*

Peter Bogetoft[§], Dan Lund Christensen[¶], Ivan Damgård[‡], Martin Geisler[‡], Thomas Jakobsen[¶], Mikkel Krøigaard [‡], Janus Dam Nielsen[‡], Jesper Buus Nielsen[‡], Kurt Nielsen[†], Jakob Pagter[¶], Michael Schwartzbach[‡], and Tomas Toft[††]

[†] Inst. of Food and Resource Economics, University of Copenhagen
[‡] Department of Computer Science, University of Aarhus
§Dept. of Economics, Copenhagen Business School
¶The Alexandra Institute
†† CWI Amsterdam and TU/e

Abstract. In this note, we report on the first large-scale and practical application of multiparty computation, which took place in January 2008. We also report on the novel cryptographic protocols that were used.

1 Introduction and History

In multiparty computation (MPC), we consider a number of players $P_1, \ldots, P_n$, who initially each hold inputs $x_1, \ldots, x_n$, and we then want to securely compute some function f on these inputs, where $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, such that P_i learns y_i but no other information. This should hold, even if players exhibit some amount of adversarial behavior. The goal can be accomplished by an interactive protocol π that the players execute. Intuitively, we want that executing π is equivalent to having a trusted party T that receives privately x_i from P_i, computes the function, and returns y_i to each P_i [1]. With such a protocol we can - in principle - solve virtually any cryptographic protocol problem. The general theory of MPC was founded in the late 80-ties [16, 3, 7]. The theory was later developed in several ways – see for instance [21, 18, 8]. An overview of the theoretical results known can be found in [6].

Despite the obvious potential that MPC has in solving a wide range of problems, we have seen virtually no practical applications of MPC in the past. This is probably in part due to the fact that direct implementation of the first general protocols would lead to very inefficient solutions. Another factor has been a general lack of understanding in the general public of the potential of the technology. A lot of research has gone into solving the efficiency problems, both for general protocols [11, 17, 9] and for special types of computations such as voting [4, 12].

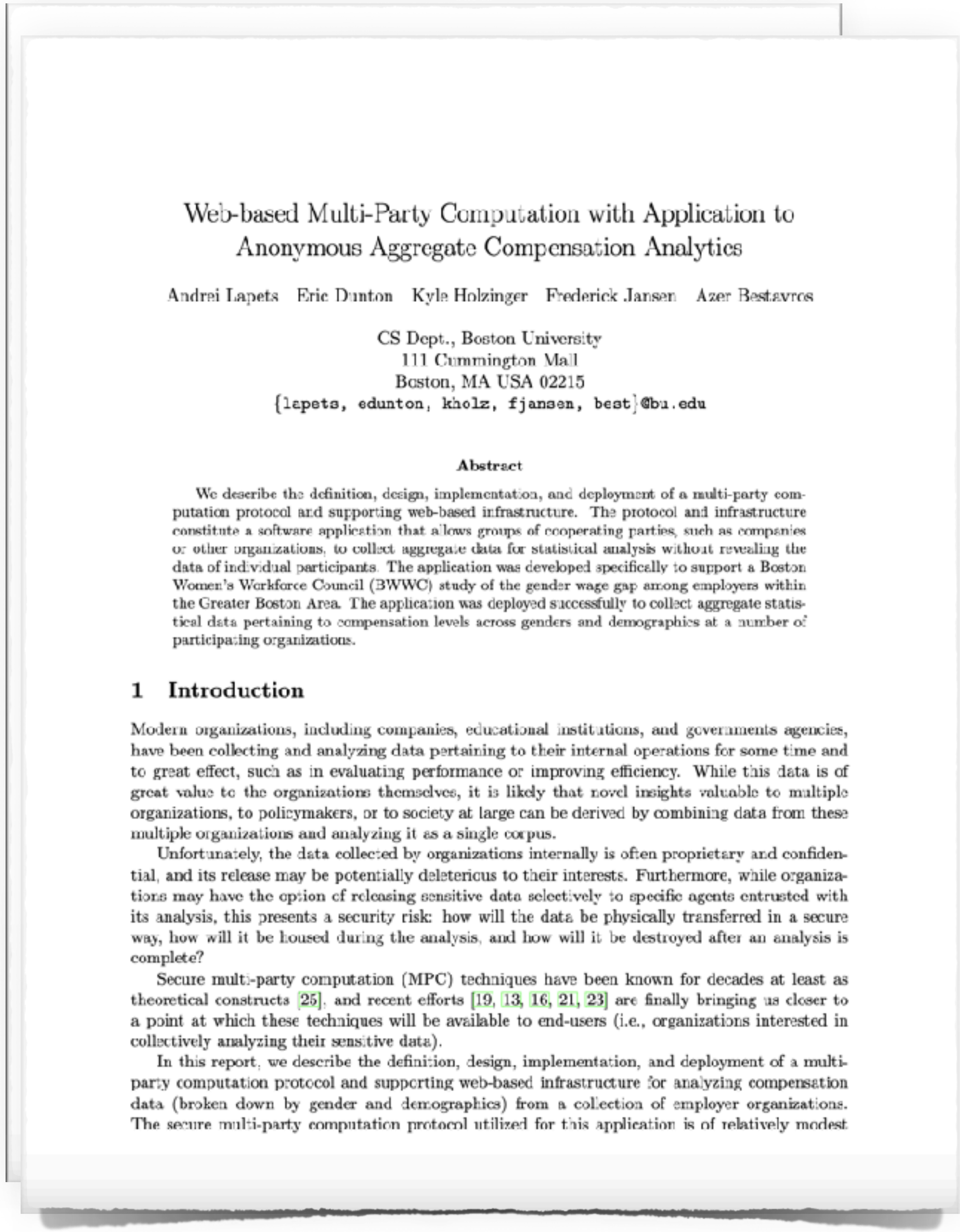
A different line of research has had explicit focus on a range of economic applications, which are particularly interesting for practical use. This approach was taken, for instance, by two research projects that the authors of this paper have been involved in: SCET (Secure Computing, Economy and Trust)[2] and SIMAP (Secure Information Management and Processing),[3], which has been responsible for the practical application of MPC described in this paper. In the economic field of mechanism design the concept of a trusted third party has been a central assumption since the 70's [15, 19, 10]. Ever since the field was initiated it has grown in momentum and turned into a truly cross disciplinary field. Today, many practical mechanisms require a trusted third party and it is natural to consider the possibility of implementing such a party using MPC. In particular, we have considered:

– Various types of auctions that involves sealed bids for different reasons. The most well-known is probably the standard highest bid auction with sealed bids, however, in terms of turnover another common variant is the so called double auction with many sellers and buyers. This auction handles scenarios where one wants to find a fair market price for a commodity given the existing supply and demand in the market.

* This work was sponsored by the Danish Strategic Research Council.
[1] This "equivalence" can be formalized using, for instance, Canetti's Universal Composability framework[5].
[2] see http://sikkerhed.alexandra.dk/uk/projects/scet
[3] see http://sikkerhed.alexandra.dk/uk/projects/simap

Secure Auctions

Privacy-preserving studies



Web-based Multi-Party Computation with Application to Anonymous Aggregate Compensation Analytics

Andrei Lapets Eric Dunton Kyle Holzinger Frederick Jansen Azer Bestavros

CS Dept., Boston University
111 Cummington Mall
Boston, MA USA 02215
{lapets, edunton, kholz, fjansen, best}@bu.edu

Abstract

We describe the definition, design, implementation, and deployment of a multi-party computation protocol and supporting web-based infrastructure. The protocol and infrastructure constitute a software application that allows groups of cooperating parties, such as companies or other organizations, to collect aggregate data for statistical analysis without revealing the data of individual participants. The application was developed specifically to support a Boston Women's Workforce Council (BWWC) study of the gender wage gap among employers within the Greater Boston Area. The application was deployed successfully to collect aggregate statistical data pertaining to compensation levels across genders and demographics at a number of participating organizations.

1 Introduction

Modern organizations, including companies, educational institutions, and governments agencies, have been collecting and analyzing data pertaining to their internal operations for some time and to great effect, such as in evaluating performance or improving efficiency. While this data is of great value to the organizations themselves, it is likely that novel insights valuable to multiple organizations, to policymakers, or to society at large can be derived by combining data from these multiple organizations and analyzing it as a single corpus.

Unfortunately, the data collected by organizations internally is often proprietary and confidential, and its release may be potentially deleterious to their interests. Furthermore, while organizations may have the option of releasing sensitive data selectively to specific agents entrusted with its analysis, this presents a security risk: how will the data be physically transferred in a secure way, how will it be housed during the analysis, and how will it be destroyed after an analysis is complete?

Secure multi-party computation (MPC) techniques have been known for decades at least as theoretical constructs [25], and recent efforts [19, 13, 16, 21, 23] are finally bringing us closer to a point at which these techniques will be available to end-users (i.e., organizations interested in collectively analyzing their sensitive data).

In this report, we describe the definition, design, implementation, and deployment of a multi-party computation protocol and supporting web-based infrastructure for analyzing compensation data (broken down by gender and demographics) from a collection of employer organizations. The secure multi-party computation protocol utilized for this application is of relatively modest

Students and Taxes: a Privacy-Preserving Social Study Using Secure Computation

Dan Bogdanov[1], Liina Kamm[1], Baldur Kubo[1], Reimo Rebane[1], Ville Sokk[1] and Riivo Talviste[1,2]

[1] Cybernetica, Tartu, Estonia
{dan.bogdanov, liina.kamm, baldur.kubo, reimo.rebane, ville.sokk, riivo.talviste}@cyber.ee
[2] University of Tartu, Institute of Computer Science, Tartu, Estonia

Abstract. We describe the use of secure multi-party computation for performing a large-scale privacy-preserving statistical study on real government data. In 2015, statisticians from the Estonian Center of Applied Research (CentAR) conducted a big data study to look for correlations between working during university studies and failing to graduate in time. The study was conducted by linking the database of individual tax payments from the Estonian Tax and Customs Board and the database of higher education events from the Ministry of Education and Research. Data collection, preparation and analysis were conducted using the Sharemind secure multi-party computation system that provided end-to-end cryptographic protection to the analysis. Using ten million tax records and half a million education records in the analysis, this is the largest cryptographically private statistical study ever conducted on real data.

Keywords: privacy, statistics, secure multi-party computation, case study

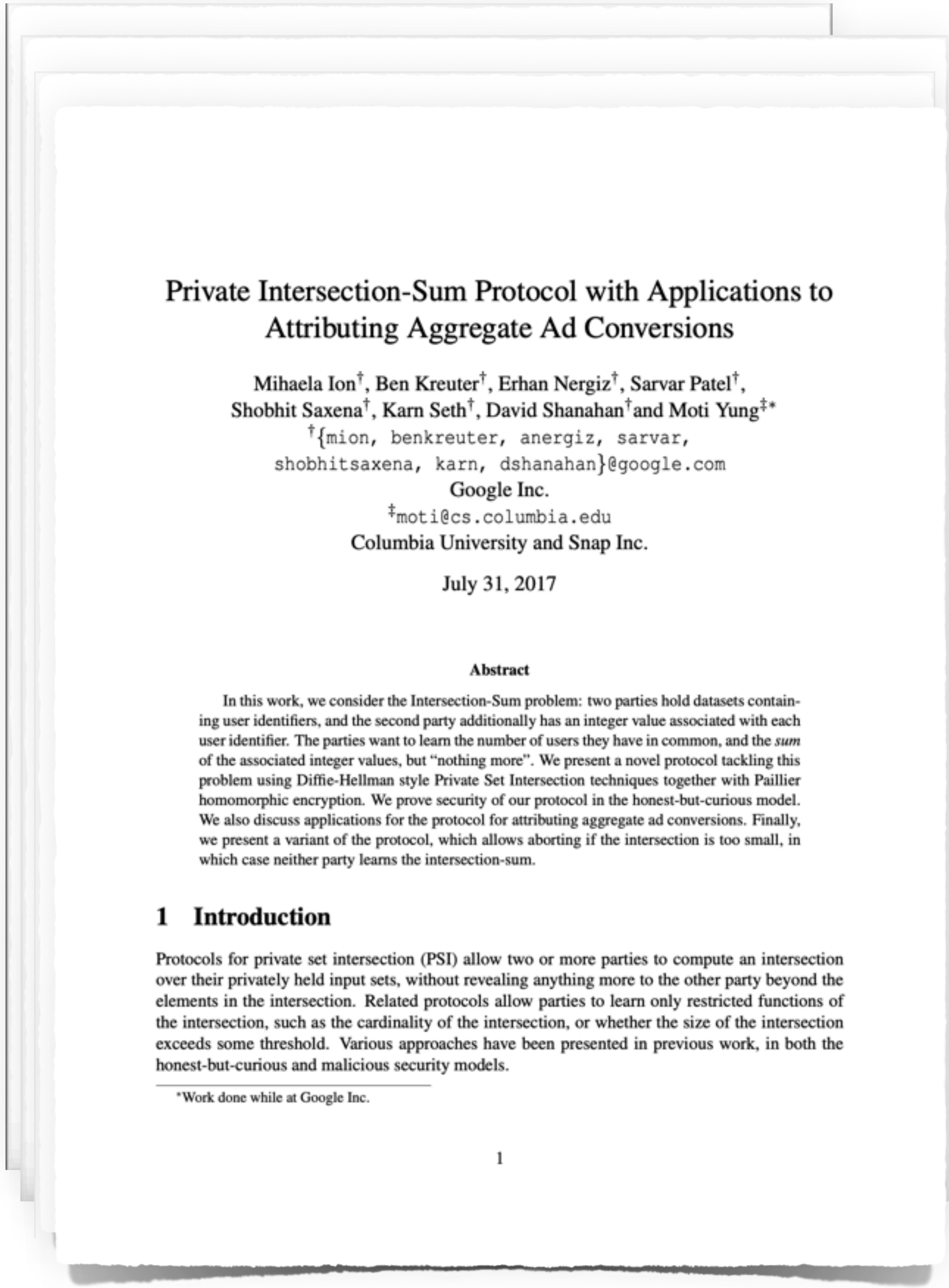
1 Introduction

Information and communication technology (ICT) is a growing industry where highly skilled specialists are in demand. This causes concern to both industry, where the wages keep rising, and the academia that cannot often match the pay grades offered by the industry. The universities in Estonia formed a hypothesis that students who work during their studies, do not graduate in the allotted time. Moreover, many students quit before graduation, thus, not acquiring the skills needed for building more complex ICT systems.

In this paper, we describe a big data study on Estonian government data that researches this topic and uses privacy-enhancing technologies to protect personal data. We collaborated with a team of social scientists who designed a statistical study that links tax and education records to determine the working habits of both ICT and non-ICT students. However, running the actual study would normally be impossible, as data protection and tax secrecy legislation

Secure Auctions

Privacy-preserving studies



Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions

Mihaela Ion[†], Ben Kreuter[†], Erhan Nergiz[†], Sarvar Patel[†], Shobhit Saxena[†], Karn Seth[†], David Shanahan[†] and Moti Yung[‡*]
[†]{mion, benkreuter, anergiz, sarvar, shobhitsaxena, karn, dshanahan}@google.com
Google Inc.
[‡]moti@cs.columbia.edu
Columbia University and Snap Inc.

July 31, 2017

Abstract

In this work, we consider the Intersection-Sum problem: two parties hold datasets containing user identifiers, and the second party additionally has an integer value associated with each user identifier. The parties want to learn the number of users they have in common, and the *sum* of the associated integer values, but "nothing more". We present a novel protocol tackling this problem using Diffie-Hellman style Private Set Intersection techniques together with Paillier homomorphic encryption. We prove security of our protocol in the honest-but-curious model. We also discuss applications for the protocol for attributing aggregate ad conversions. Finally, we present a variant of the protocol, which allows aborting if the intersection is too small, in which case neither party learns the intersection-sum.

1 Introduction

Protocols for private set intersection (PSI) allow two or more parties to compute an intersection over their privately held input sets, without revealing anything more to the other party beyond the elements in the intersection. Related protocols allow parties to learn only restricted functions of the intersection, such as the cardinality of the intersection, or whether the size of the intersection exceeds some threshold. Various approaches have been presented in previous work, in both the honest-but-curious and malicious security models.

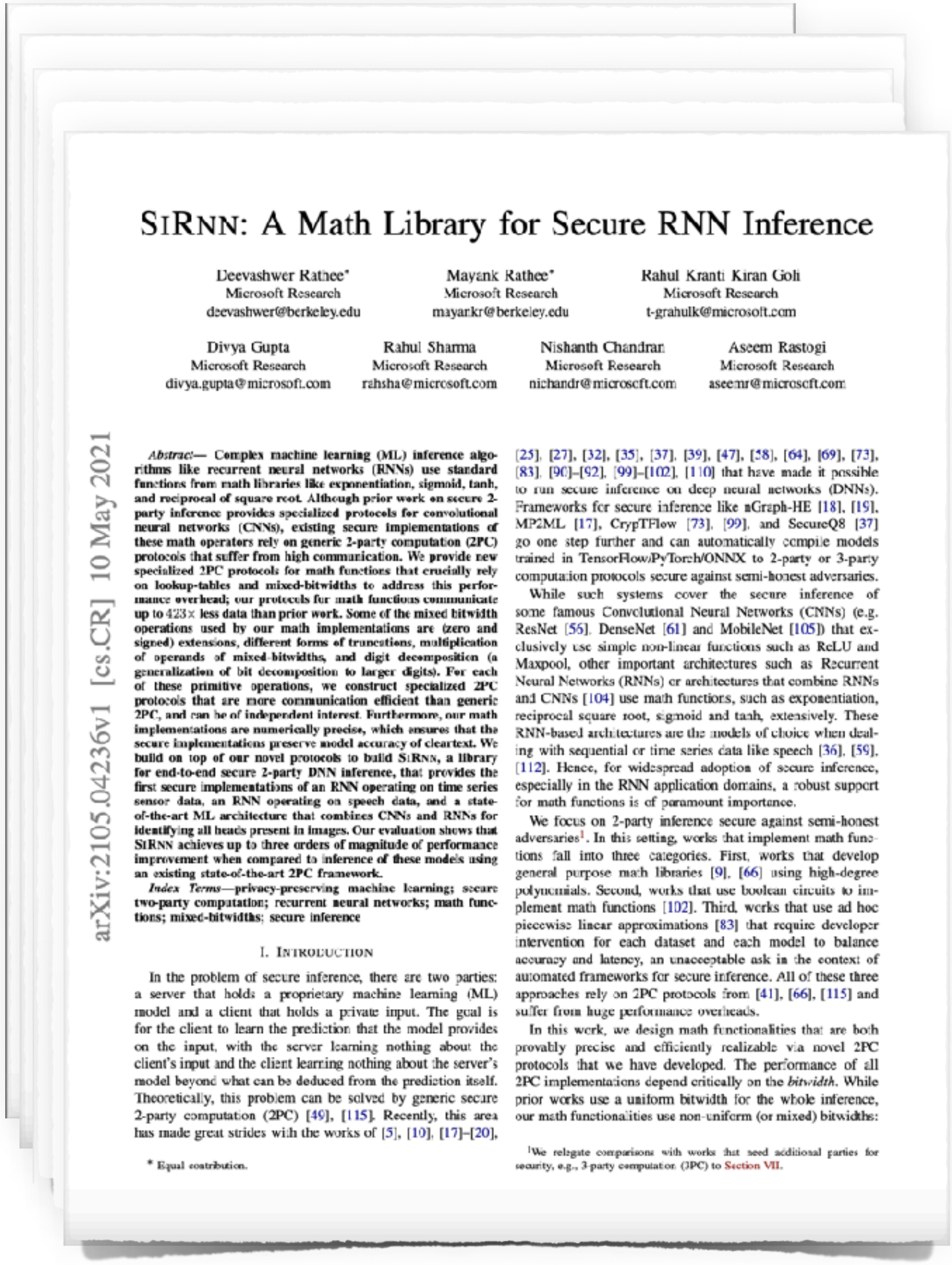
*Work done while at Google Inc.

1

Secure Auctions

Privacy-preserving studies

Privacy-preserving advertising



SIRNN: A Math Library for Secure RNN Inference

Deevashwer Rathee*
Microsoft Research
deevashwer@berkeley.edu

Mayank Rathee*
Microsoft Research
mayankr@berkeley.edu

Rahul Kranti Kiran Goli
Microsoft Research
t-grahulk@microsoft.com

Divya Gupta
Microsoft Research
divya.gupta@microsoft.com

Rahul Sharma
Microsoft Research
rahsha@microsoft.com

Nishanth Chandran
Microsoft Research
nichandr@microsoft.com

Aseem Rastogi
Microsoft Research
aseemr@microsoft.com

arXiv:2105.04236v1 [cs.CR] 10 May 2021

Abstract— Complex machine learning (ML) inference algorithms like recurrent neural networks (RNNs) use standard functions from math libraries like exponentiation, sigmoid, tanh, and reciprocal of square root. Although prior work on secure 2-party inference provides specialized protocols for convolutional neural networks (CNNs), existing secure implementations of these math operators rely on generic 2-party computation (2PC) protocols that suffer from high communication. We provide new specialized 2PC protocols for math functions that crucially rely on lookup-tables and mixed-bitwidths to address this performance overhead; our protocols for math functions communicate up to 423× less data than prior work. Some of the mixed bitwidth operations used by our math implementations are (zero and signed) extensions, different forms of truncations, multiplication of operands of mixed-bitwidths, and digit decomposition (a generalization of bit decomposition to larger digits). For each of these primitive operations, we construct specialized 2PC protocols that are more communication efficient than generic 2PC, and can be of independent interest. Furthermore, our math implementations are numerically precise, which ensures that the secure implementations preserve model accuracy of cleartext. We build on top of our novel protocols to build SIRNN, a library for end-to-end secure 2-party DNN inference, that provides the first secure implementations of an RNN operating on time series sensor data, an RNN operating on speech data, and a state-of-the-art ML architecture that combines CNNs and RNNs for identifying all heads present in images. Our evaluation shows that SIRNN achieves up to three orders of magnitude of performance improvement when compared to inference of these models using an existing state-of-the-art 2PC framework.

Index Terms—privacy-preserving machine learning; secure two-party computation; recurrent neural networks; math functions; mixed-bitwidths; secure inference

I. Introduction

In the problem of secure inference, there are two parties: a server that holds a proprietary machine learning (ML) model and a client that holds a private input. The goal is for the client to learn the prediction that the model provides on the input, with the server learning nothing about the client's input and the client learning nothing about the server's model beyond what can be deduced from the prediction itself. Theoretically, this problem can be solved by generic secure 2-party computation (2PC) [49], [115]. Recently, this area has made great strides with the works of [5], [10], [17]–[20], [25], [27], [32], [35], [37], [39], [47], [58], [64], [69], [73], [83], [90]–[92], [99]–[102], [110] that have made it possible to run secure inference on deep neural networks (DNNs). Frameworks for secure inference like nGraph-HE [18], [19], MP2ML [17], CrypTFlow [73], [99], and SecureQ8 [37] go one step further and can automatically compile models trained in TensorFlow/PyTorch/ONNX to 2-party or 3-party computation protocols secure against semi-honest adversaries.

While such systems cover the secure inference of some famous Convolutional Neural Networks (CNNs) (e.g. ResNet [56], DenseNet [61] and MobileNet [105]) that exclusively use simple non-linear functions such as ReLU and Maxpool, other important architectures such as Recurrent Neural Networks (RNNs) or architectures that combine RNNs and CNNs [104] use math functions, such as exponentiation, reciprocal square root, sigmoid and tanh, extensively. These RNN-based architectures are the models of choice when dealing with sequential or time series data like speech [36], [59], [112]. Hence, for widespread adoption of secure inference, especially in the RNN application domains, a robust support for math functions is of paramount importance.

We focus on 2-party inference secure against semi-honest adversaries[1]. In this setting, works that implement math functions fall into three categories. First, works that develop general purpose math libraries [9], [66] using high-degree polynomials. Second, works that use boolean circuits to implement math functions [102]. Third, works that use ad hoc piecewise linear approximations [83] that require developer intervention for each dataset and each model to balance accuracy and latency, an unacceptable ask in the context of automated frameworks for secure inference. All of these three approaches rely on 2PC protocols from [41], [66], [115] and suffer from huge performance overheads.

In this work, we design math functionalities that are both provably precise and efficiently realizable via novel 2PC protocols that we have developed. The performance of all 2PC implementations depend critically on the *bitwidth*. While prior works use a uniform bitwidth for the whole inference, our math functionalities use non-uniform (or mixed) bitwidths:

* Equal contribution.

[1]We relegate comparisons with works that need additional parties for security, e.g., 3-party computation (3PC) to Section VII.

Secure Auctions

Privacy-preserving studies

Privacy-preserving advertising

Privacy-preserving analytics
(*Secure Machine Learning*)

Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications

David Byrd
db@gatech.edu
School of Interactive Computing
Georgia Institute of Technology
Atlanta, Georgia

Antigoni Polychroniadou
antigoni.poly@jpmorgan.com
J.P. Morgan AI Research
New York, New York

ABSTRACT

Federated Learning enables a population of clients, working with a trusted server, to collaboratively learn a shared machine learning model while keeping each client's data within its own local systems. This reduces the risk of exposing sensitive data, but it is still possible to reverse engineer information about a client's private data set from communicated model parameters. Most federated learning systems therefore use differential privacy to introduce noise to the parameters. This adds uncertainty to any attempt to reveal private client data, but also reduces the accuracy of the shared model, limiting the useful scale of privacy-preserving noise. A system can further reduce the coordinating server's ability to recover private client information, without additional accuracy loss, by also including secure multiparty computation. An approach combining both techniques is especially relevant to financial firms as it allows new possibilities for collaborative learning without exposing sensitive client data. This could produce more accurate models for important tasks like optimal trade execution, credit origination, or fraud detection. The key contributions of this paper are: We present a privacy-preserving federated learning protocol to a non-specialist audience, demonstrate it using logistic regression on a real-world credit card fraud data set, and evaluate it using an open-source simulation platform which we have adapted for the development of federated learning systems.

KEYWORDS

federated learning, simulation, multiagent, finance, privacy

ACM Reference Format:
David Byrd and Antigoni Polychroniadou. 2020. Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications. In *ACM International Conference on AI in Finance (ICAIF '20), October 15–16, 2020, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3383455.3422562

ICAIF '20, October 15–16, 2020, New York, NY, USA

ACM ISBN 978-1-4503-7584-9/20/10...$15.00
https://doi.org/10.1145/3383455.3422562

1 INTRODUCTION

Modern financial firms routinely need to conduct analysis of large data sets stored across multiple servers or devices. A typical response is to combine these data sets into a single central database, but this approach introduces a number of privacy challenges: The institution may not have appropriate authority or permission to transfer locally stored information, the owner of the data may not *want* it shared, and centralization of the data may worsen the potential consequences of a data breach.

For example, the mobile app ai.type collected personal data from its users' phones and uploaded this information to a central database. Security researchers gained access to the database and obtained the names, email addresses, passwords, and other sensitive information of 31 million users of the Android version of the app. Such incidents highlight the risks and challenges associated with centralized data solutions. [5]

In this section, we motivate our approach while providing an extensive non-technical overview of the underlying techniques.

1.1 Federated Learning

One approach to mitigate the mentioned privacy concerns is to analyze the multiple data sets separately and share only the resulting insights from each analysis. This approach is realized in a recently-introduced technique called federated analysis. [2] Federated *learning*, already adopted by large companies like Google, allows users to share insights (perhaps the parameters of a trained model) from the data on their laptops or mobile devices *without* ever sharing the data itself, typically as follows:

1. Users train a local model on their individual data.
2. Each user sends their model weights to a trusted server.
3. The server computes an average-weight shared model.
4. The shared model is returned to all of the users.
5. Users retrain a local model starting from the shared model.

For instance, email providers could use federated learning to reduce the amount of spam their customers receive. Instead of each provider using its own spam filter trained from its customers' reported spam email, the providers could combine their models to create a shared spam-detection mechanism, without sharing their individual customers' reported spam emails. For a survey of recent advances in federated learning, see Kairouz et al. [13]

It is still possible, however, for a malicious party to potentially compromise the privacy of the individual users by inferring details of a training data set from the trained model's weights or parameters [16, 19]. It is important to protect sensitive user information while still providing highly accurate inferences.

Secure Auctions

Privacy-preserving studies

Privacy-preserving advertising

Privacy-preserving analytics

(Secure Machine Learning)

Financial Fraud Detection

Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications

David Byrd
db@gatech.edu
School of Interactive Computing
Georgia Institute of Technology
Atlanta, Georgia

Antigoni Polychroniadou
antigoni.poly@jpmorgan.com
J.P. Morgan AI Research
New York, New York

ABSTRACT

Federated Learning enables a population of clients, working with a trusted server, to collaboratively learn a shared machine learning model while keeping each client's data within its own local systems. This reduces the risk of exposing sensitive data, but it is still possible to reverse engineer information about a client's private data set from communicated model parameters. Most federated learning systems therefore use differential privacy to introduce noise to the parameters. This adds uncertainty to any attempt to reveal private client data, but also reduces the accuracy of the shared model, limiting the useful scale of privacy-preserving noise. A system can further reduce the coordinating server's ability to recover private client information, without additional accuracy loss, by also including secure multiparty computation. An approach combining both techniques is especially relevant to financial firms as it allows new possibilities for collaborative learning without exposing sensitive client data. This could produce more accurate models for important tasks like optimal trade execution, credit origination, or fraud detection. The key contributions of this paper are: We present a privacy-preserving federated learning protocol to a non-specialist audience, demonstrate it using logistic regression on a real-world credit card fraud data set, and evaluate it using an open-source simulation platform which we have adapted for the development of federated learning systems.

KEYWORDS

federated learning, simulation, multiagent, finance, privacy

ACM Reference Format:
David Byrd and Antigoni Polychroniadou. 2020. Differentially Private Secure Multi-Party Computation for Federated Learning in Financial Applications. In *ACM International Conference on AI in Finance (ICAIF '20), October 15–16, 2020, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3383455.3422562

ICAIF '20, October 15–16, 2020, New York, NY, USA

ACM ISBN 978-1-4503-7584-9/20/10...$15.00
https://doi.org/10.1145/3383455.3422562

1 INTRODUCTION

Modern financial firms routinely need to conduct analysis of large data sets stored across multiple servers or devices. A typical response is to combine these data sets into a single central database, but this approach introduces a number of privacy challenges: The institution may not have appropriate authority or permission to transfer locally stored information, the owner of the data may not *want* it shared, and centralization of the data may worsen the potential consequences of a data breach.

For example, the mobile app ai.type collected personal data from its users' phones and uploaded this information to a central database. Security researchers gained access to the database and obtained the names, email addresses, passwords, and other sensitive information of 31 million users of the Android version of the app. Such incidents highlight the risks and challenges associated with centralized data solutions. [5]

In this section, we motivate our approach while providing an extensive non-technical overview of the underlying techniques.

1.1 Federated Learning

One approach to mitigate the mentioned privacy concerns is to analyze the multiple data sets separately and share only the resulting insights from each analysis. This approach is realized in a recently-introduced technique called federated analysis. [2] Federated *learning*, already adopted by large companies like Google, allows users to share insights (perhaps the parameters of a trained model) from the data on their laptops or mobile devices *without* ever sharing the data itself, typically as follows:

1. Users train a local model on their individual data.
2. Each user sends their model weights to a trusted server.
3. The server computes an average-weight shared model
4. The shared model is returned to all of the users.
5. Users retrain a local model starting from the shared model.

For instance, email providers could use federated learning to reduce the amount of spam their customers receive. Instead of each provider using its own spam filter trained from its customers' reported spam email, the providers could combine their models to create a shared spam-detection mechanism, without sharing their individual customers' reported spam emails. For a survey of recent advances in federated learning, see Kairouz et al. [13]

It is still possible, however, for a malicious party to potentially compromise the privacy of the individual users by inferring details of a training data set from the trained model's weights or parameters [16, 19]. It is important to protect sensitive user information while still providing highly accurate inferences.

Secure Auctions

Privacy-preserving studies

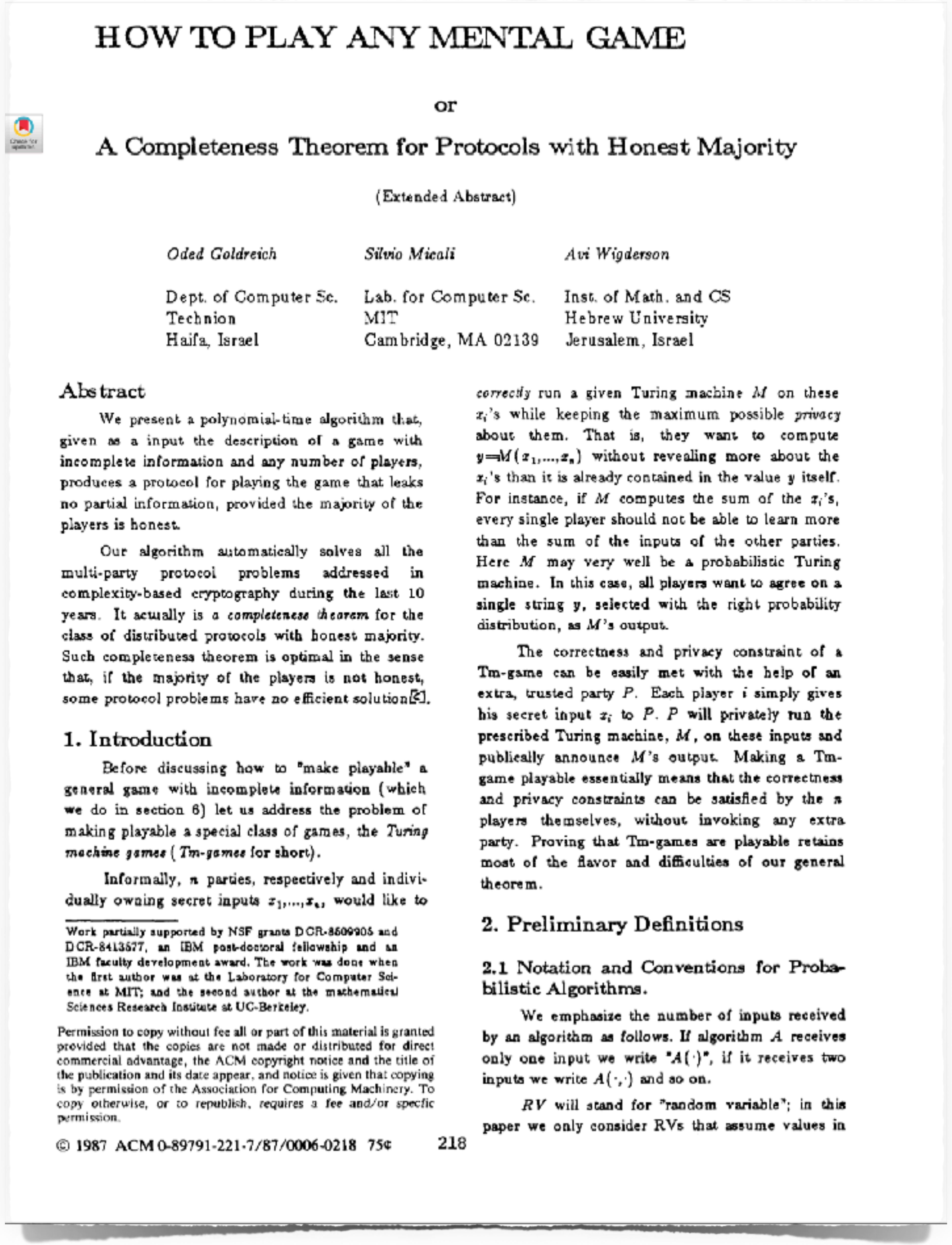
Privacy-preserving advertising

Privacy-preserving analytics
(Secure Machine Learning)

Financial Fraud Detection

…and much more

Classic GMW Protocol



HOW TO PLAY ANY MENTAL GAME

or

A Completeness Theorem for Protocols with Honest Majority

(Extended Abstract)

Oded Goldreich — Dept. of Computer Sc., Technion, Haifa, Israel

Silvio Micali — Lab. for Computer Sc., MIT, Cambridge, MA 02139

Avi Wigderson — Inst. of Math. and CS, Hebrew University, Jerusalem, Israel

Abstract

We present a polynomial-time algorithm that, given as a input the description of a game with incomplete information and any number of players, produces a protocol for playing the game that leaks no partial information, provided the majority of the players is honest.

Our algorithm automatically solves all the multi-party protocol problems addressed in complexity-based cryptography during the last 10 years. It actually is *a completeness theorem* for the class of distributed protocols with honest majority. Such completeness theorem is optimal in the sense that, if the majority of the players is not honest, some protocol problems have no efficient solution[C].

1. Introduction

Before discussing how to "make playable" a general game with incomplete information (which we do in section 6) let us address the problem of making playable a special class of games, the *Turing machine games* (*Tm-games* for short).

Informally, n parties, respectively and individually owning secret inputs $x_1,\ldots,x_n$, would like to *correctly* run a given Turing machine M on these x_i's while keeping the maximum possible *privacy* about them. That is, they want to compute $y=M(x_1,\ldots,x_n)$ without revealing more about the x_i's than it is already contained in the value y itself. For instance, if M computes the sum of the x_i's, every single player should not be able to learn more than the sum of the inputs of the other parties. Here M may very well be a probabilistic Turing machine. In this case, all players want to agree on a single string y, selected with the right probability distribution, as M's output.

The correctness and privacy constraint of a Tm-game can be easily met with the help of an extra, trusted party P. Each player i simply gives his secret input x_i to P. P will privately run the prescribed Turing machine, M, on these inputs and publically announce M's output. Making a Tm-game playable essentially means that the correctness and privacy constraints can be satisfied by the n players themselves, without invoking any extra party. Proving that Tm-games are playable retains most of the flavor and difficulties of our general theorem.

2. Preliminary Definitions

2.1 Notation and Conventions for Probabilistic Algorithms.

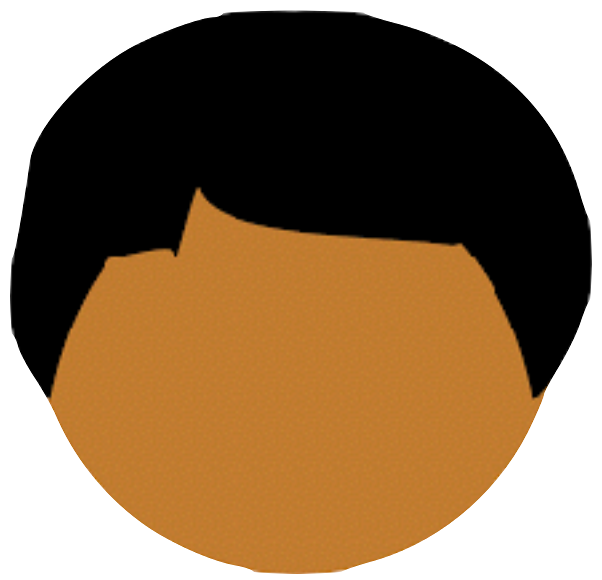
We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write "$A(\cdot)$", if it receives two inputs we write $A(\cdot,\cdot)$ and so on.

RV will stand for "random variable"; in this paper we only consider RVs that assume values in

Work partially supported by NSF grants DCR-8609905 and DCR-8413577, an IBM post-doctoral fellowship and an IBM faculty development award. The work was done when the first author was at the Laboratory for Computer Science at MIT; and the second author at the mathematical Sciences Research Institute at UC-Berkeley.

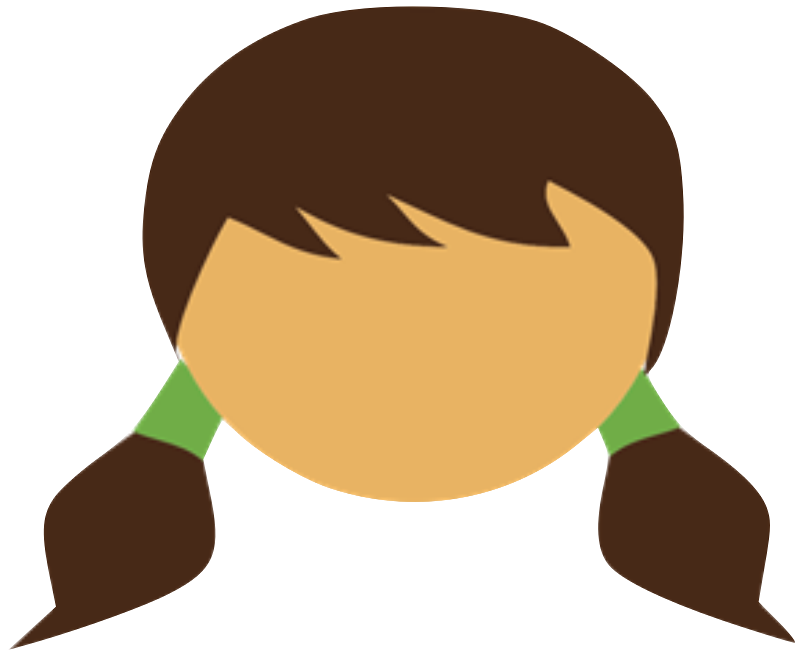
How to run any program…

For parties that are **honest but curious (semi-honest)**



Sender

1-out-of-2 Oblivious Transfer



Receiver


m_0, m_1
1-out-of-2 Oblivious Transfer
Sender
Receiver


m_0, m_1
$b \in \{0,1\}$
Sender
1-out-of-2 Oblivious Transfer
Receiver

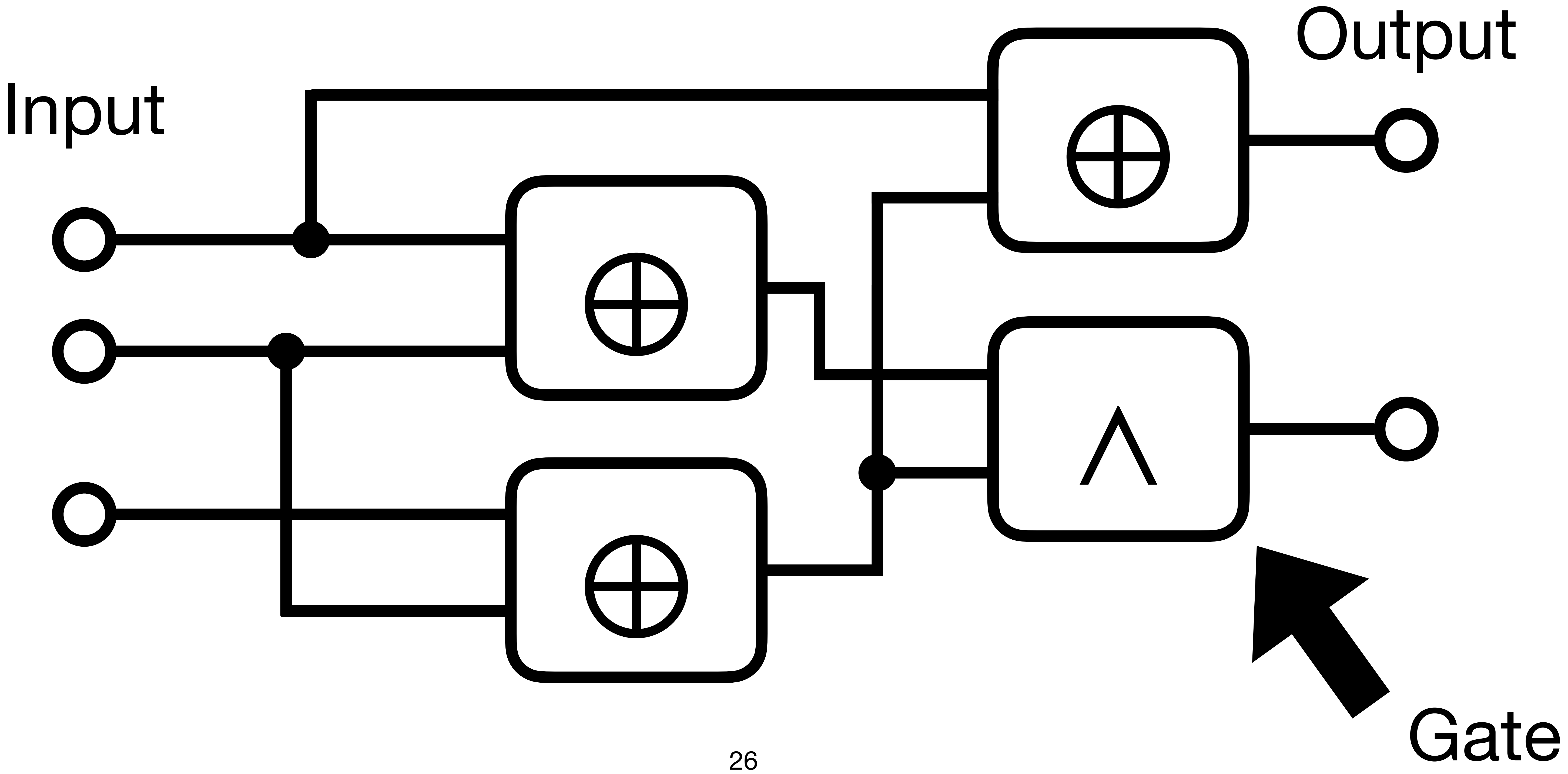

m_0, m_1
$b \in \{0,1\}$
Sender
1-out-of-2
Oblivious
Transfer
Receiver
m_b


m_0, m_1
$b \in \{0,1\}$
Sender
1-out-of-2 Oblivious Transfer
Receiver
$\perp$
m_b

OT is a standard cryptographic primitive, and there are many protocols that implement it

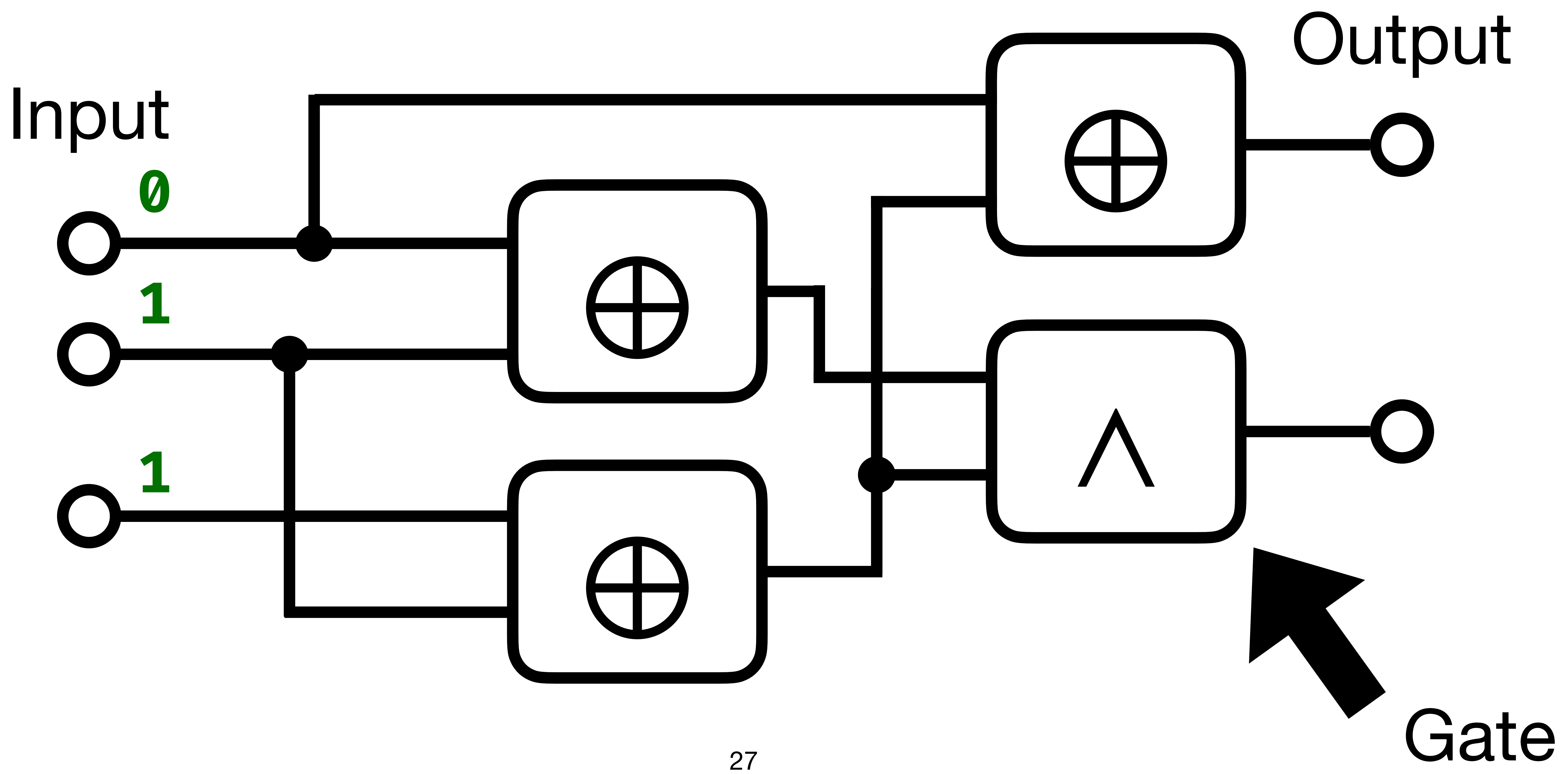
A Boolean Circuit *is a directed acyclic graph where*

- *Each node has* fan-in *two (and unbounded fan-out).*
- *Each node has a label* $\wedge$ *or* $\oplus$
- *There are two distinguished wires labelled 0 and 1*



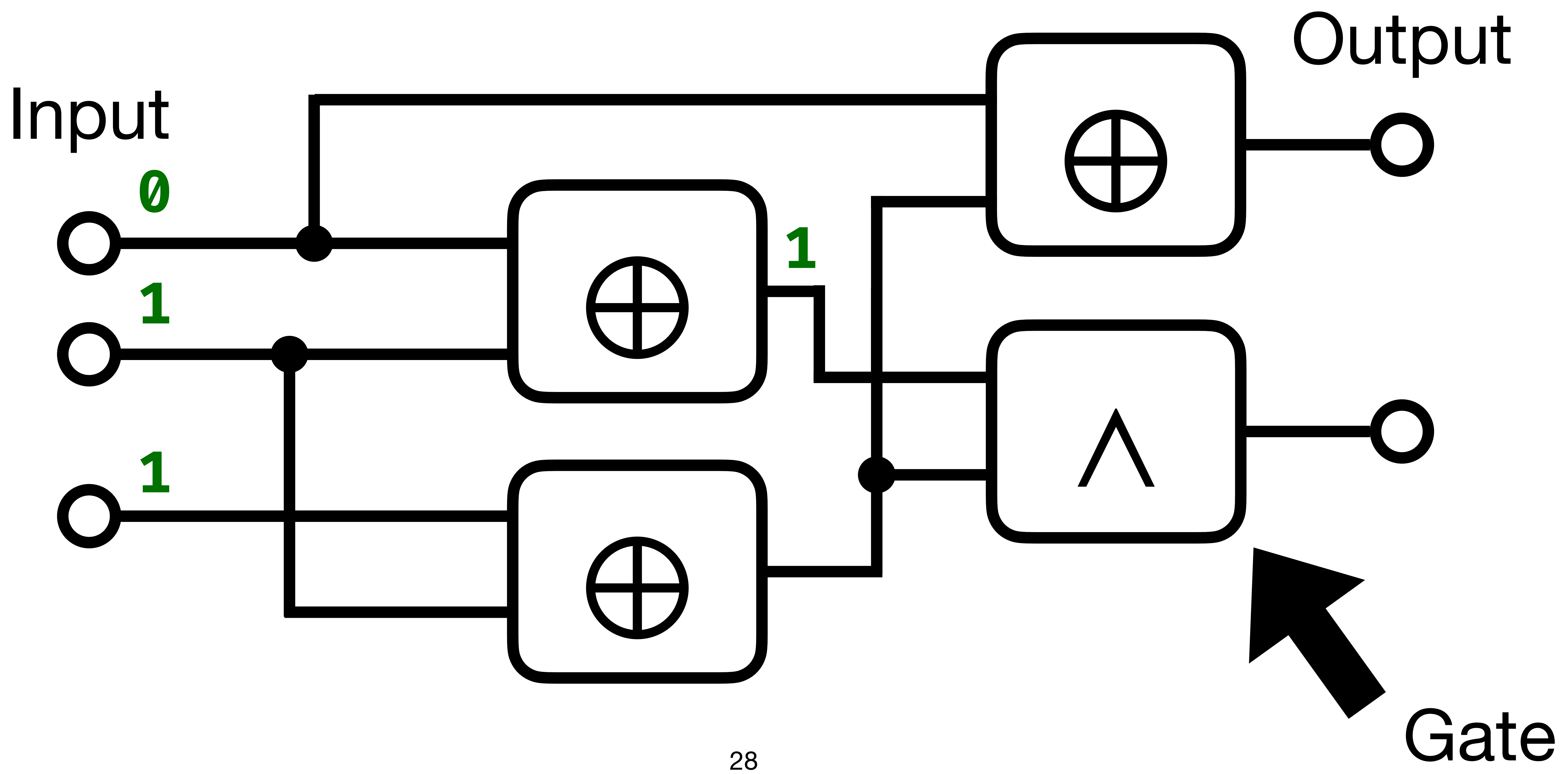
A Boolean Circuit *is a directed acyclic graph where*

- *Each node has* fan-in *two (and unbounded fan-out).*
- *Each node has a label* $\wedge$ *or* $\oplus$
- *There are two distinguished wires labelled 0 and 1*



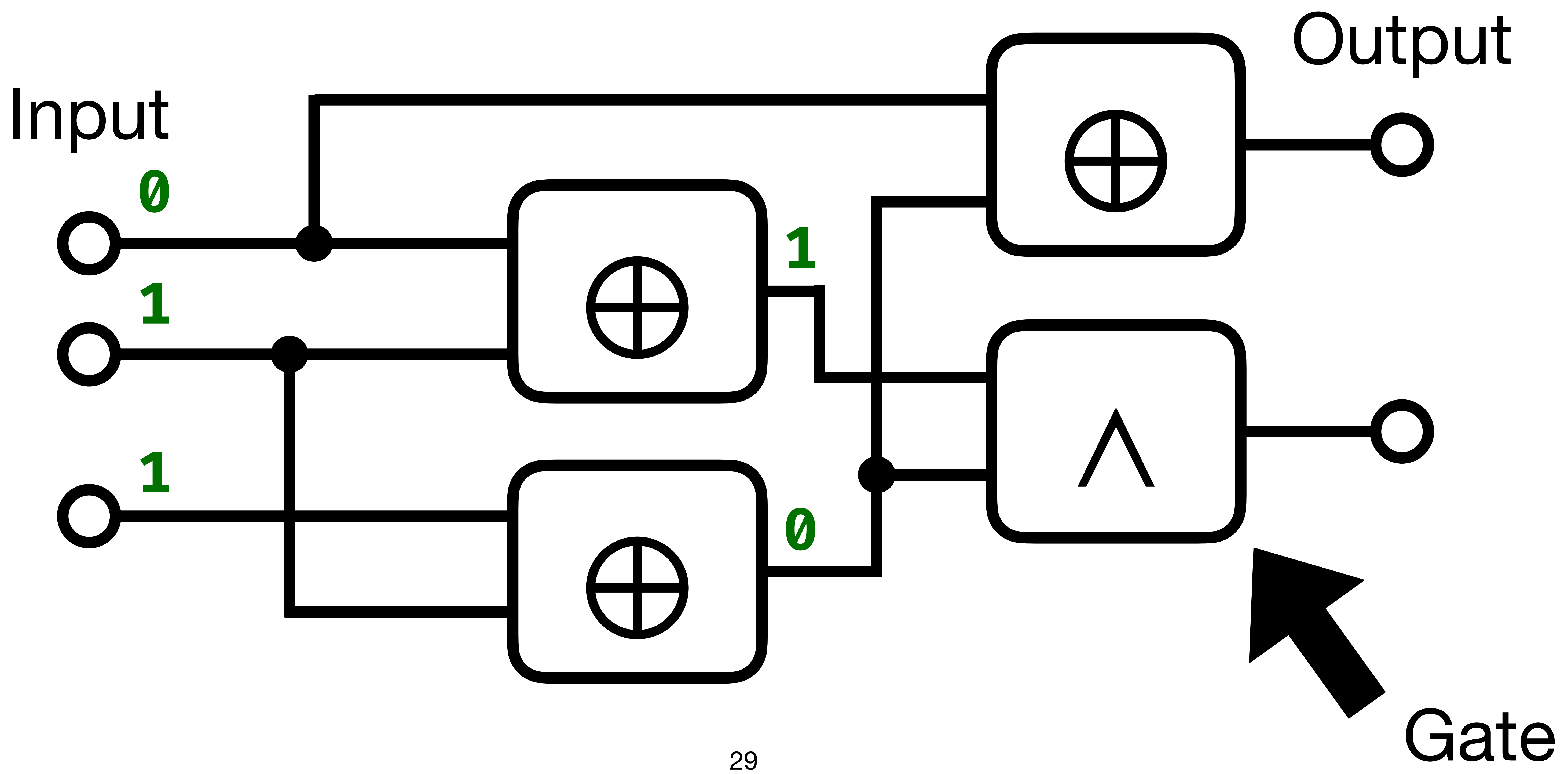
A Boolean Circuit *is a directed acyclic graph where*

- *Each node has* fan-in *two (and unbounded fan-out).*
- *Each node has a label* $\wedge$ *or* $\oplus$
- *There are two distinguished wires labelled 0 and 1*



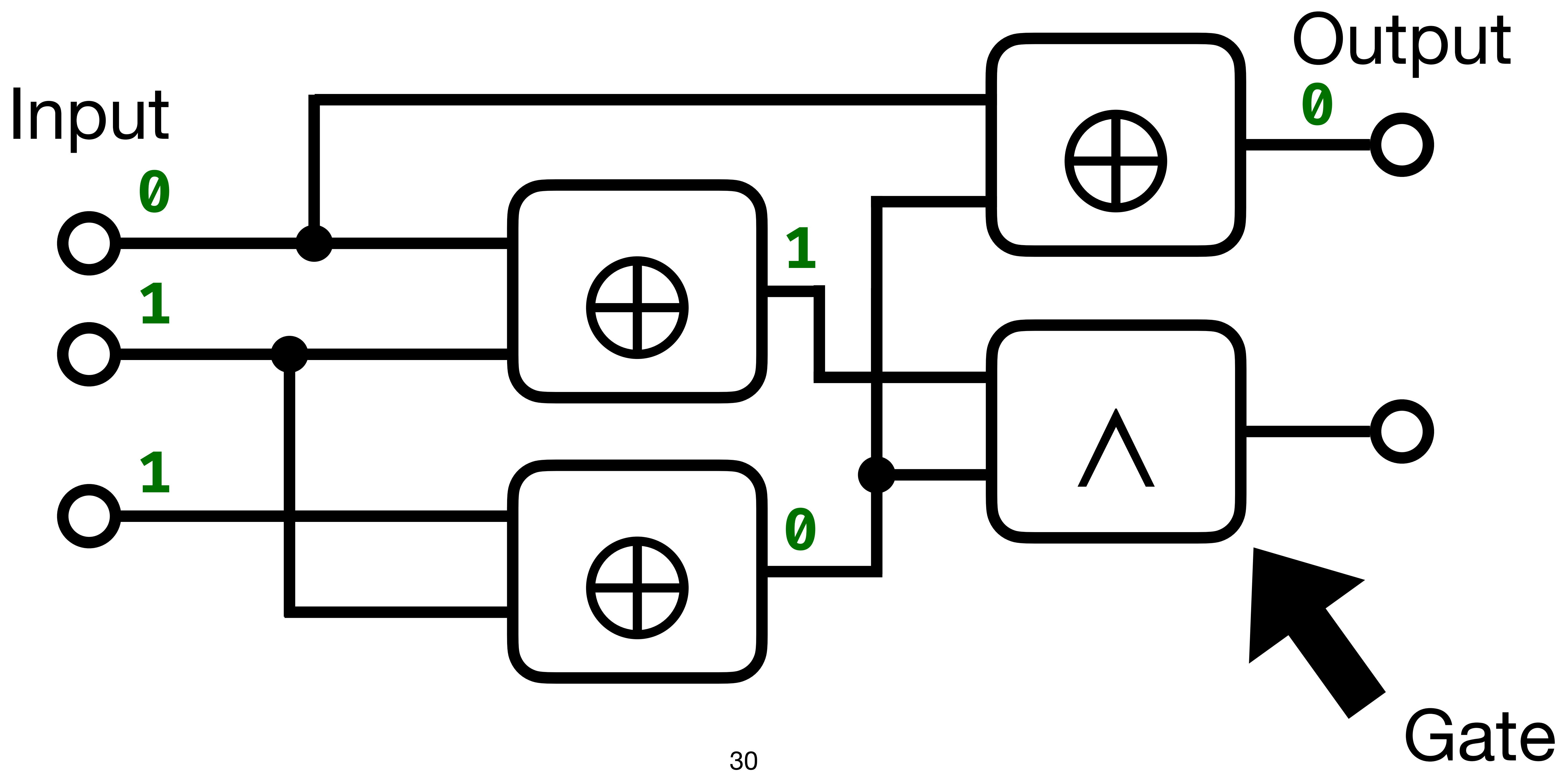
A Boolean Circuit *is a directed acyclic graph where*

- *Each node has* fan-in *two (and unbounded fan-out).*
- *Each node has a label* $\wedge$ *or* $\oplus$
- *There are two distinguished wires labelled 0 and 1*



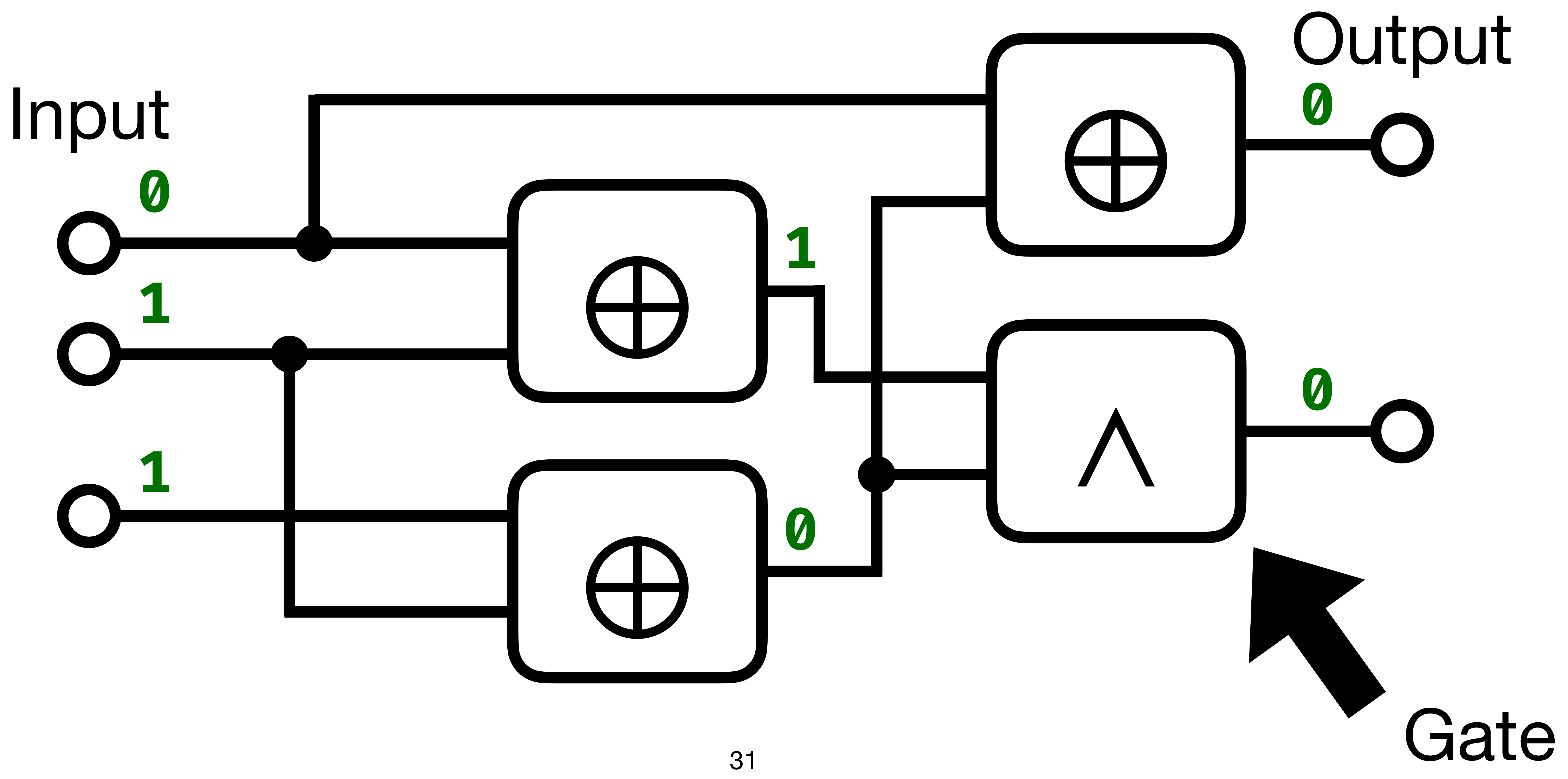
A Boolean Circuit *is a directed acyclic graph where*

- *Each node has* fan-in *two (and unbounded fan-out).*
- *Each node has a label* $\wedge$ *or* $\oplus$
- *There are two distinguished wires labelled 0 and 1*



A Boolean Circuit *is a directed acyclic graph where*

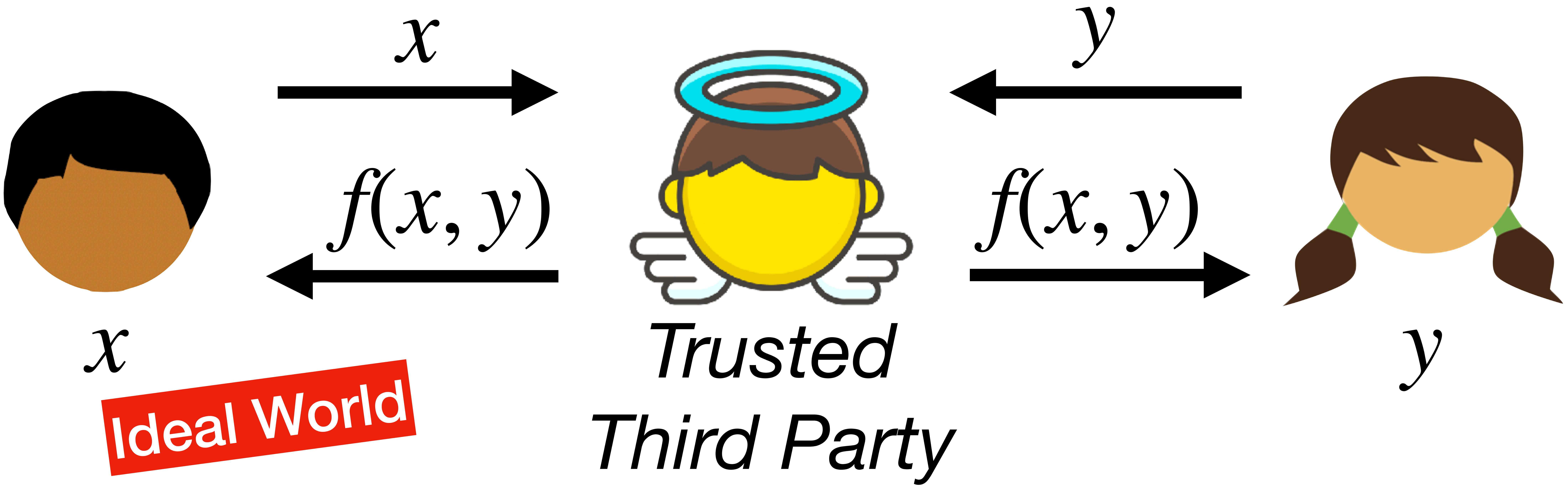
- *Each node has* fan-in *two (and unbounded fan-out).*
- *Each node has a label* $\wedge$ *or* $\oplus$
- *There are two distinguished wires labelled 0 and 1*



Fact: $\{\wedge, \oplus, 1\}$ *is a complete Boolean basis.*

For any Boolean function $f: \{0,1\}^n \to \{0,1\}^m$*, there exists a Boolean circuit over* $\{\wedge, \oplus, 1\}$ *that computes* f*.*

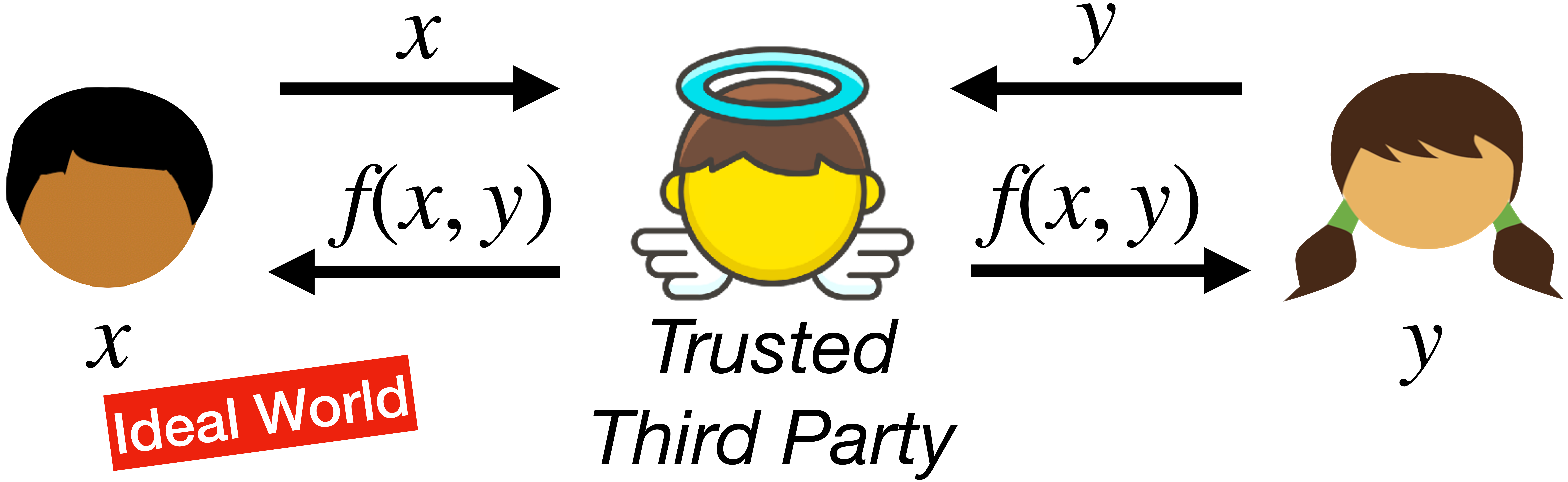
I.e., Boolean circuits can compute any bounded function



Ideal World

GMW Protocol

Real World



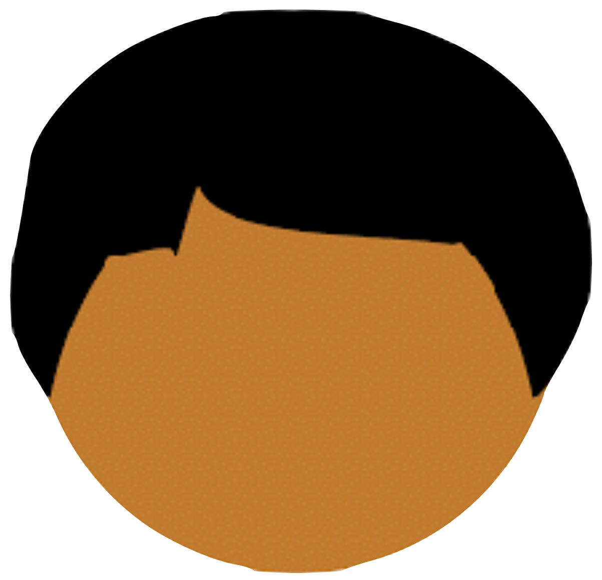
GMW Protocol

*Hint: Use **a lot** of Oblivious Transfer*

Real World

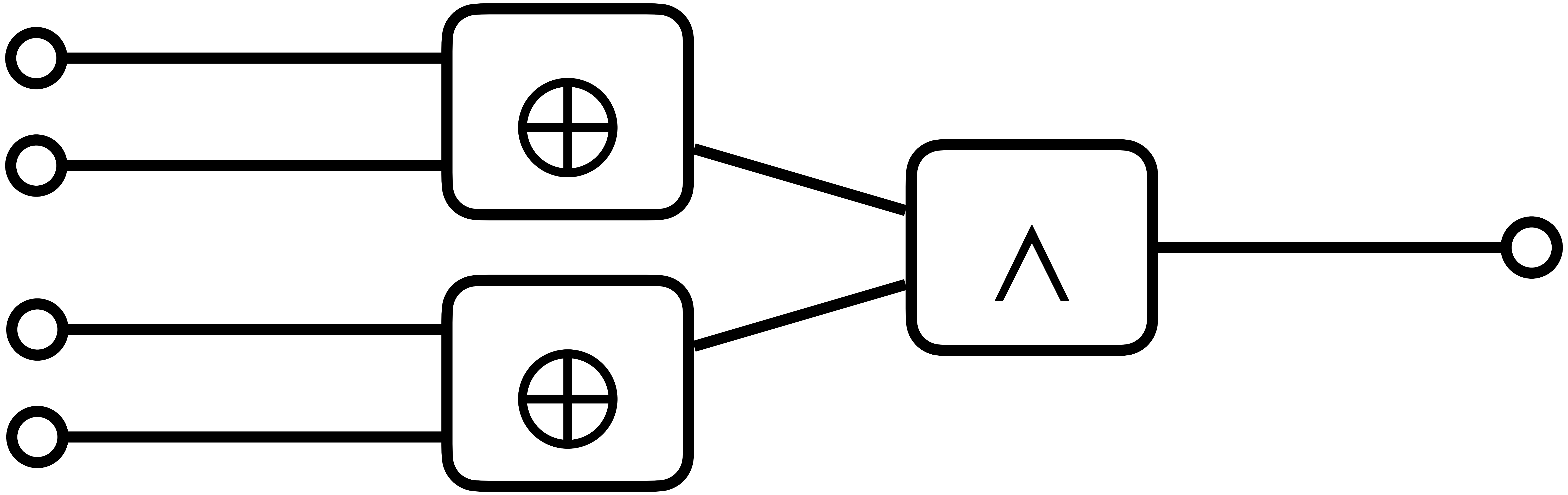
Step 1 of GMW:

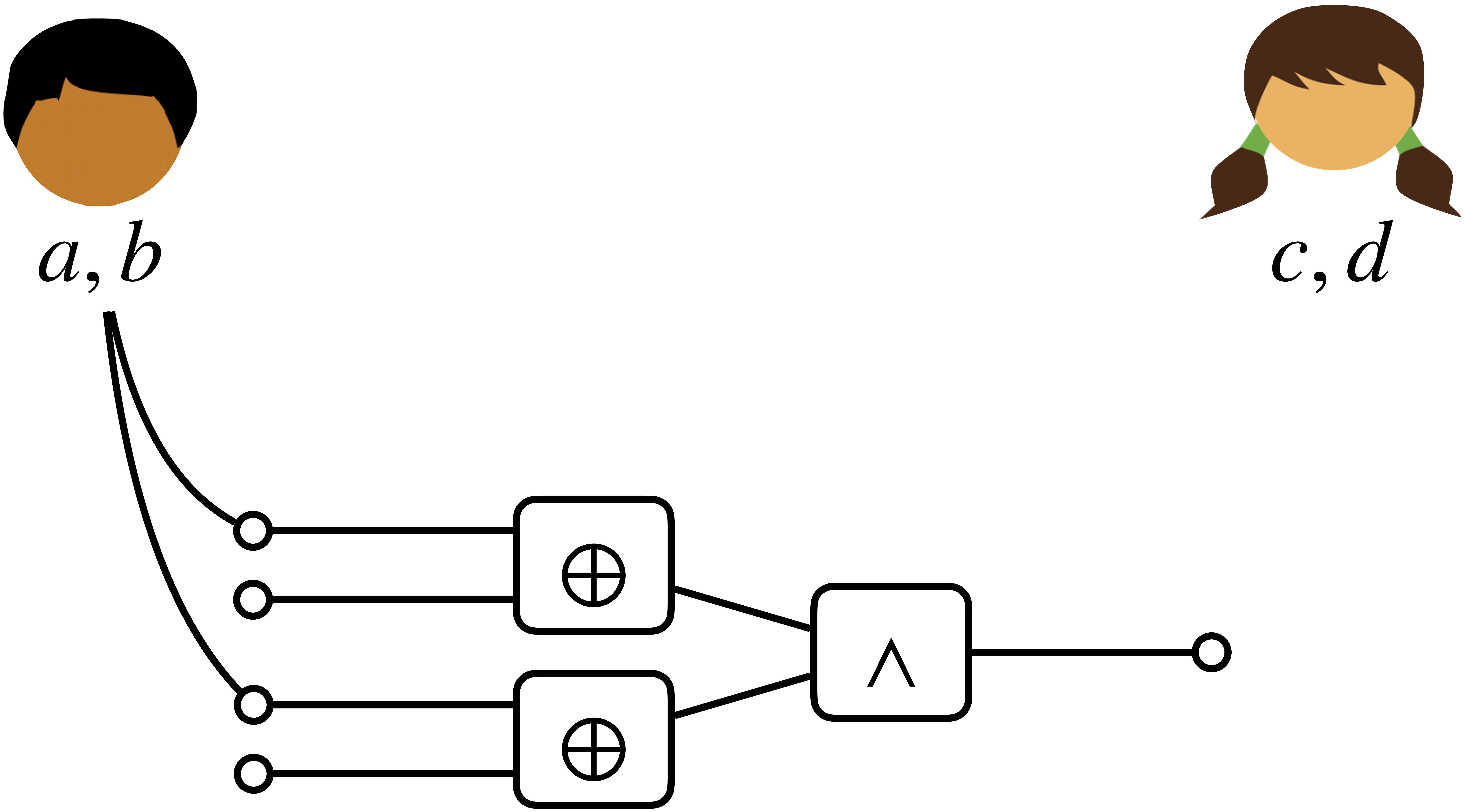
Express program F as a Boolean circuit C

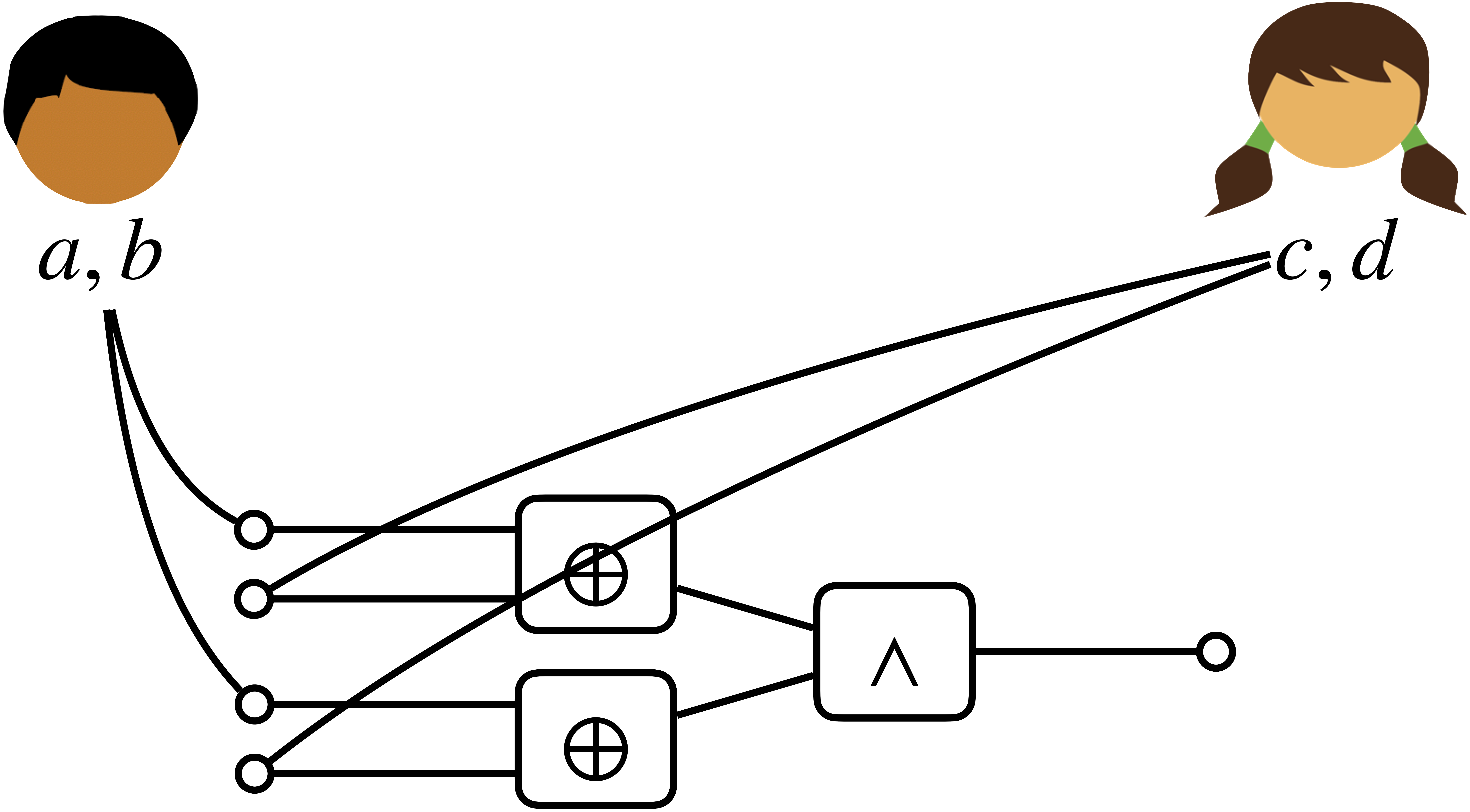


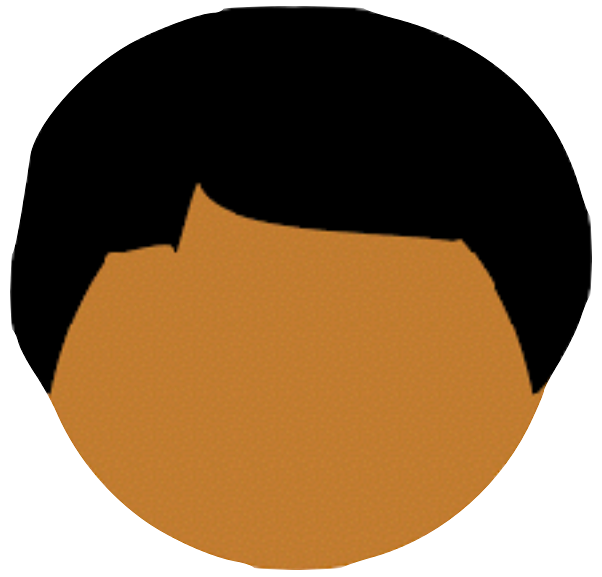
a, b

c, d

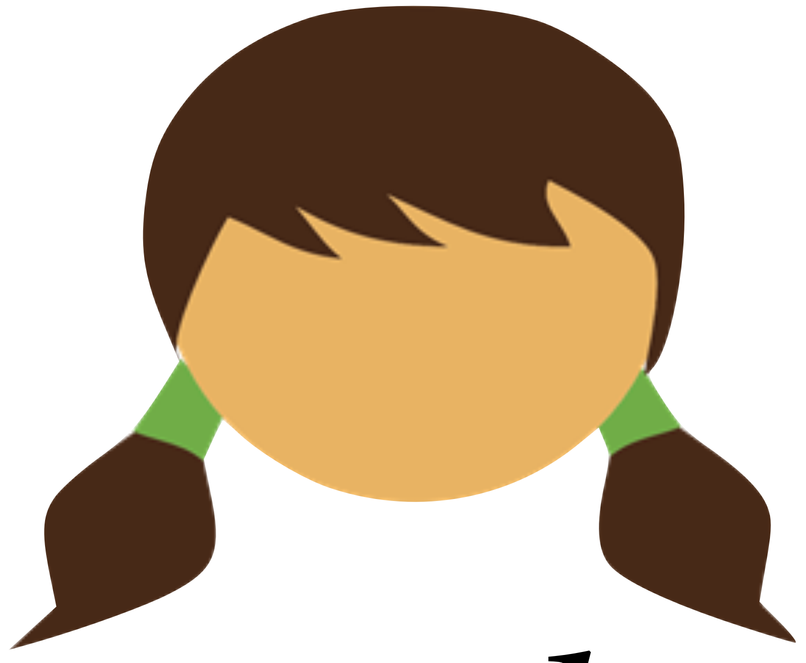



a, b
c, d
$\oplus$
$\oplus$
$\wedge$

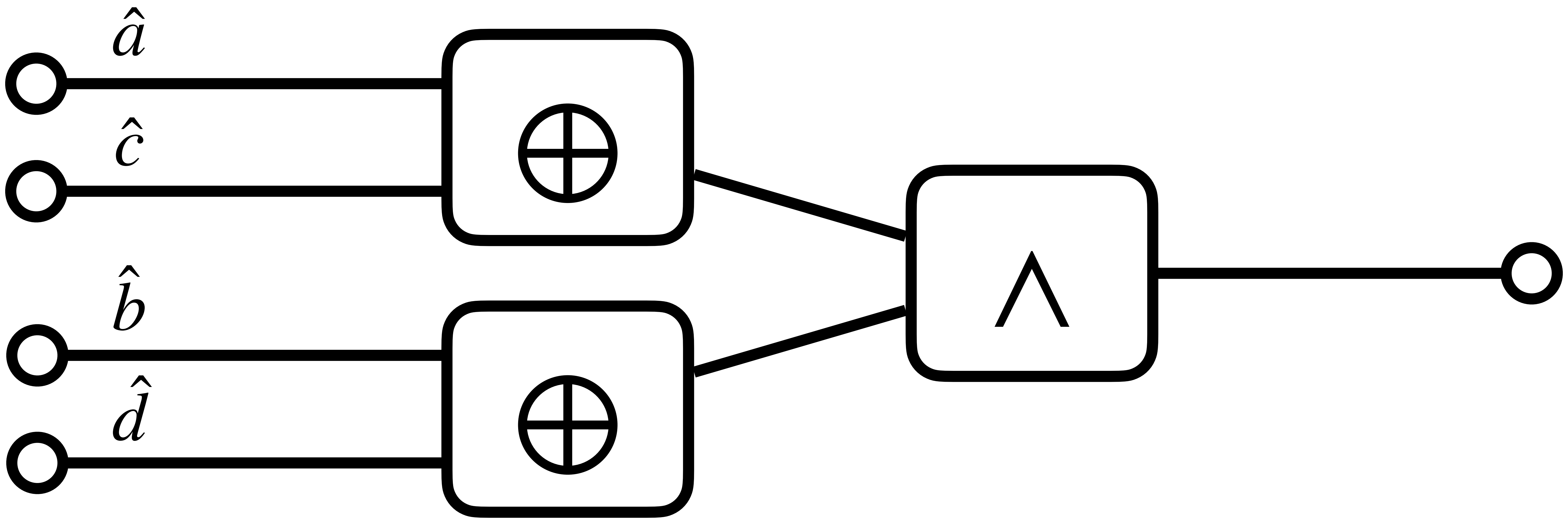

a, b
c, d
$\oplus$
$\oplus$
$\wedge$

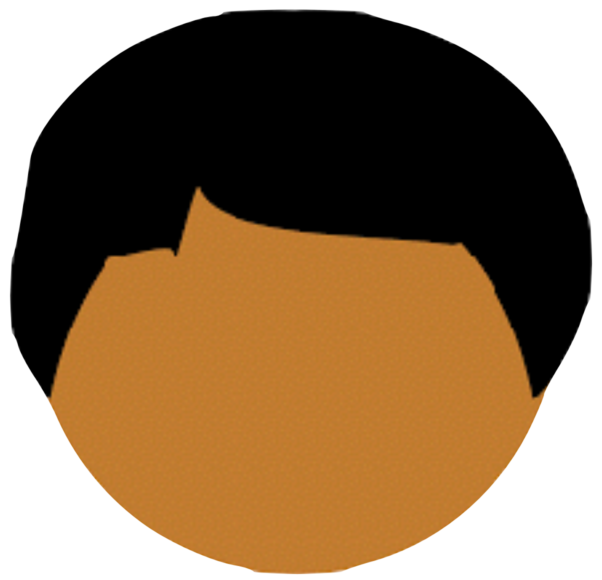


a, b

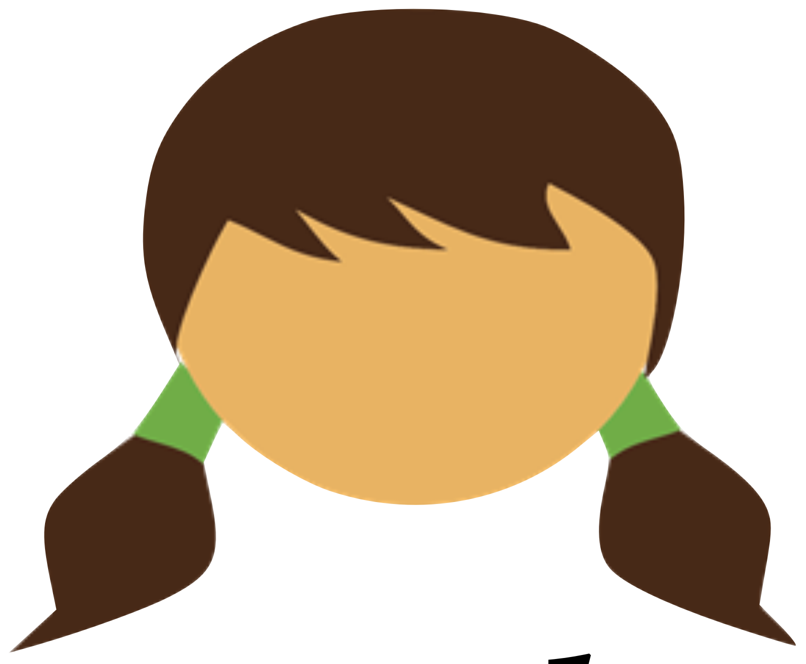


c, d

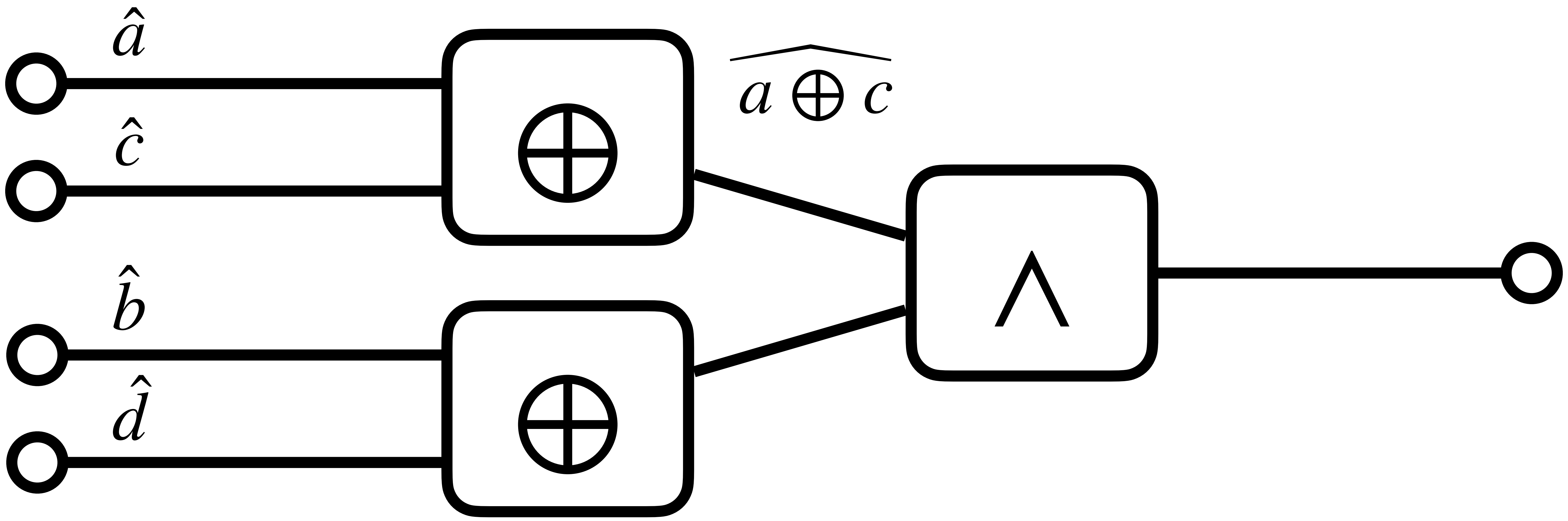


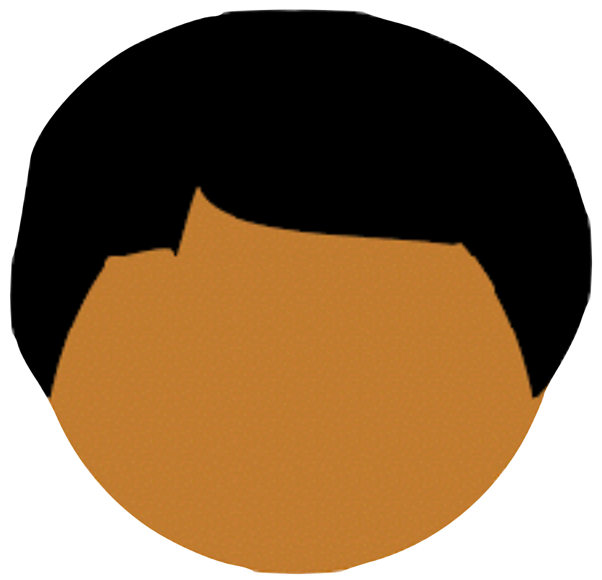


a, b

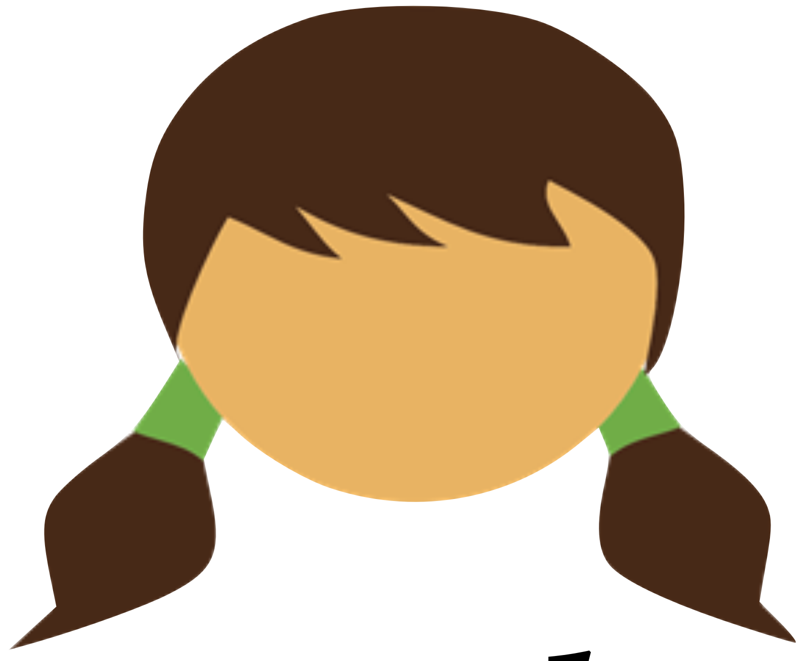


c, d

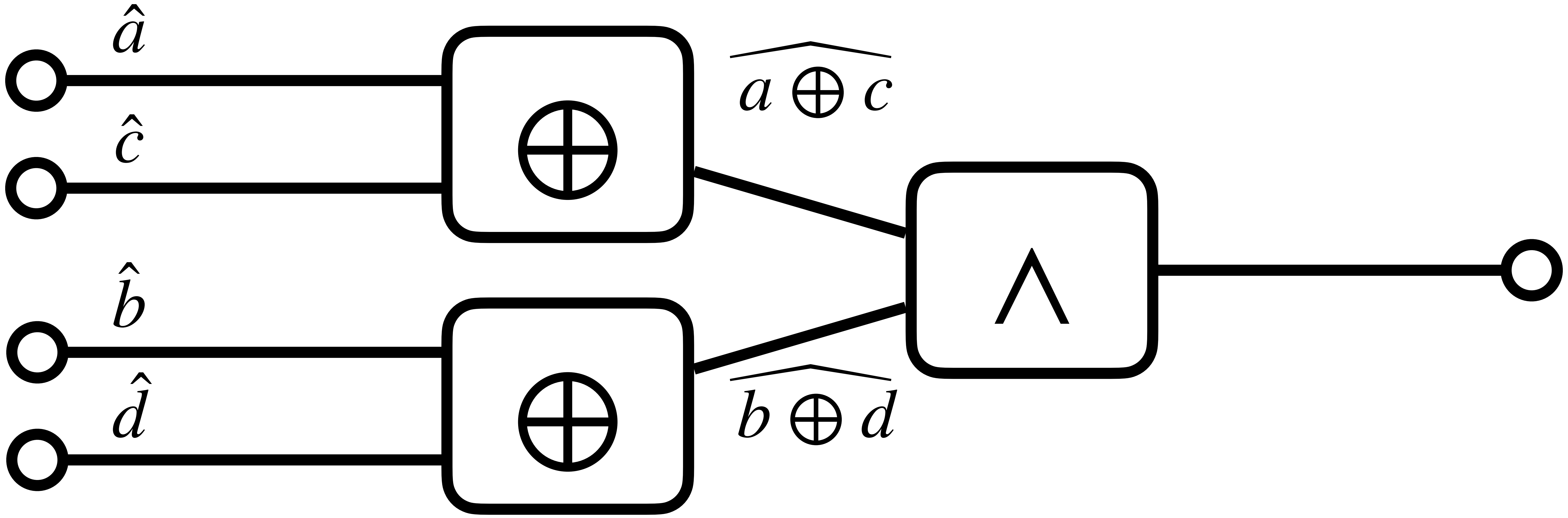




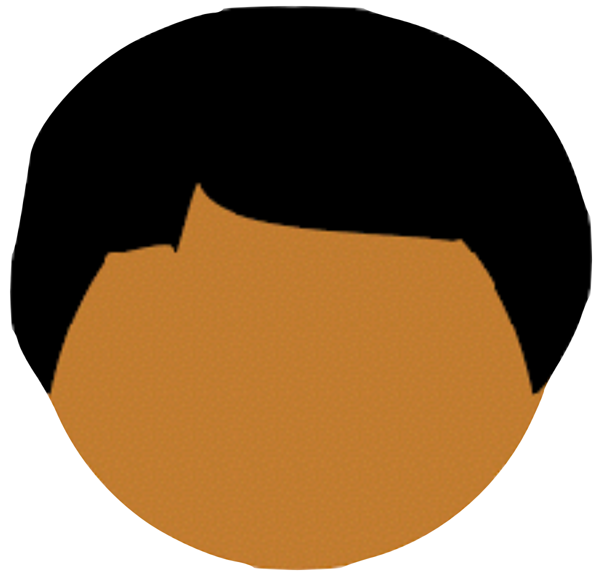
a, b



c, d

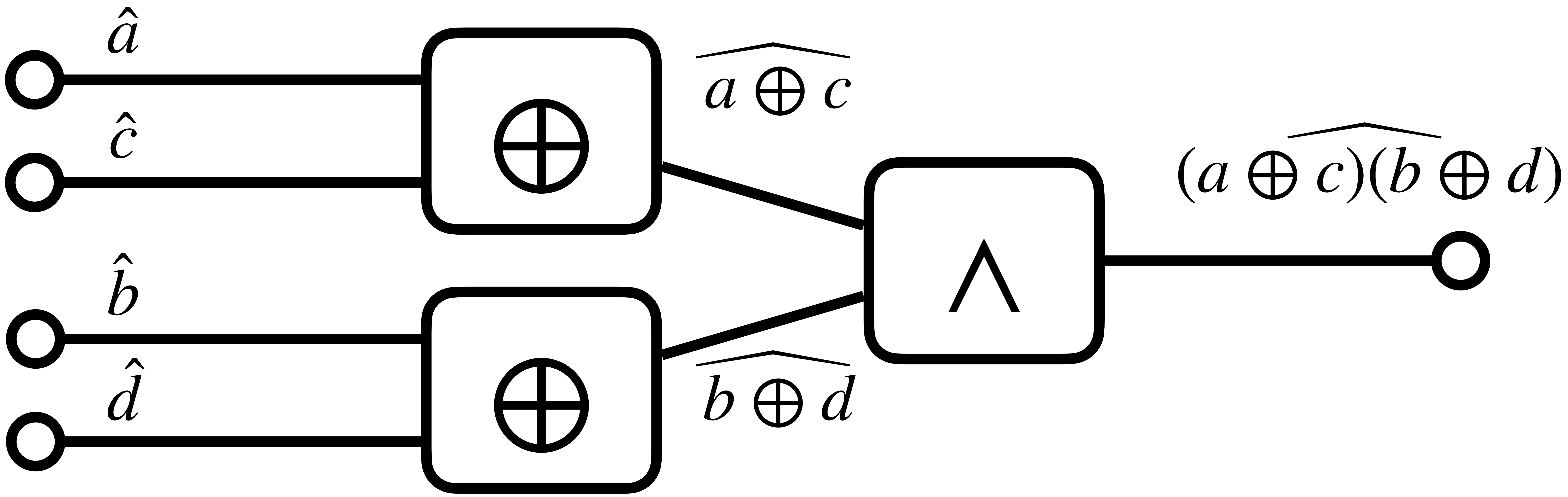


$\hat{a}$
$\hat{c}$
$\hat{b}$
$\hat{d}$
$\oplus$
$\oplus$
$\widehat{a \oplus c}$
$\widehat{b \oplus d}$
$\wedge$



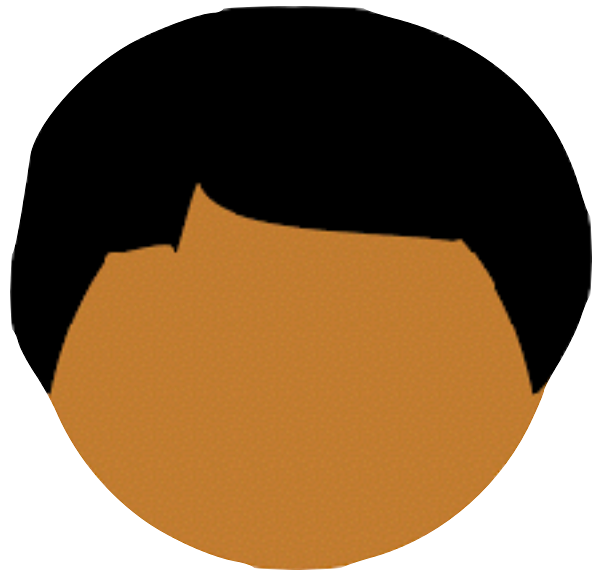
a, b

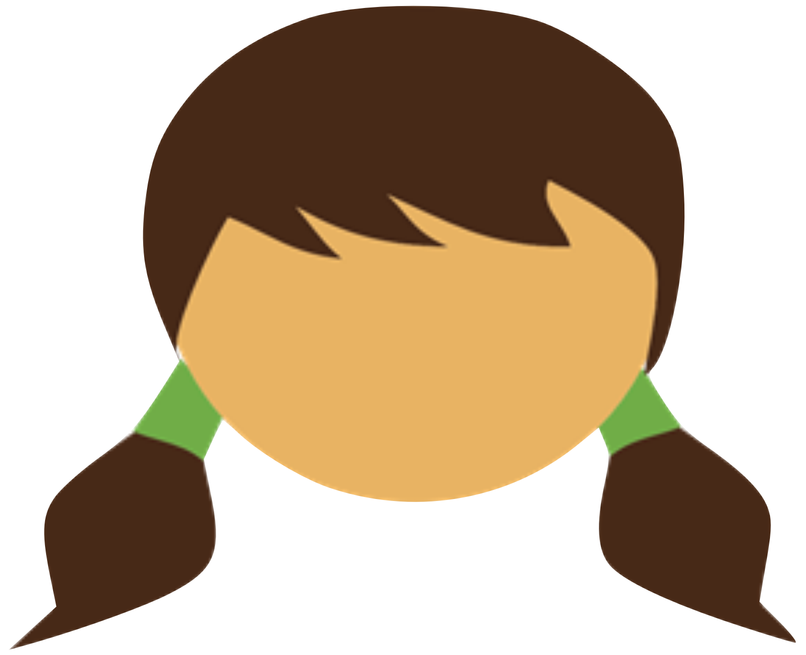
c, d



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

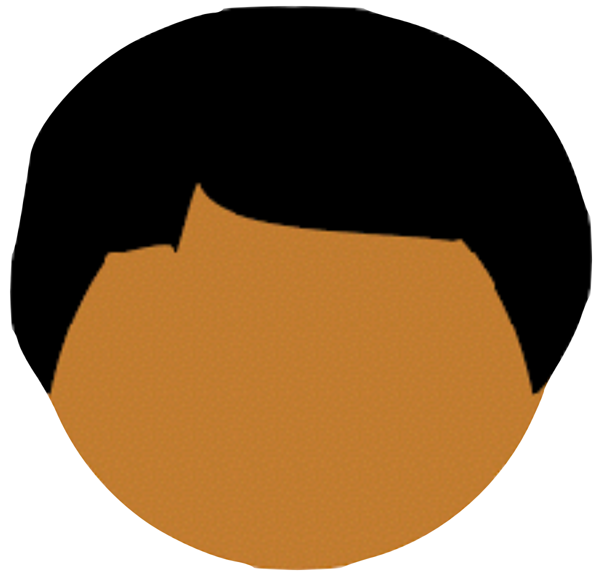
XOR Secret Shares

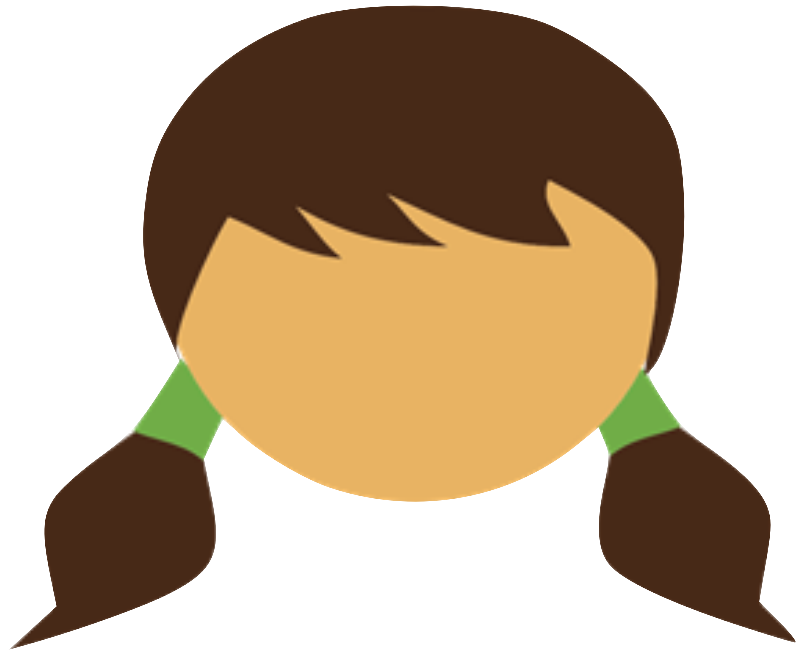




The XOR secret sharing *of a bit* x *is a pair of bits* $\langle x_0, x_1 \rangle$ *where* P_0 *holds* x_0 *and* P_1 *holds* x_1*, and where* $x_0 \oplus x_1 = x$

XOR Secret Shares

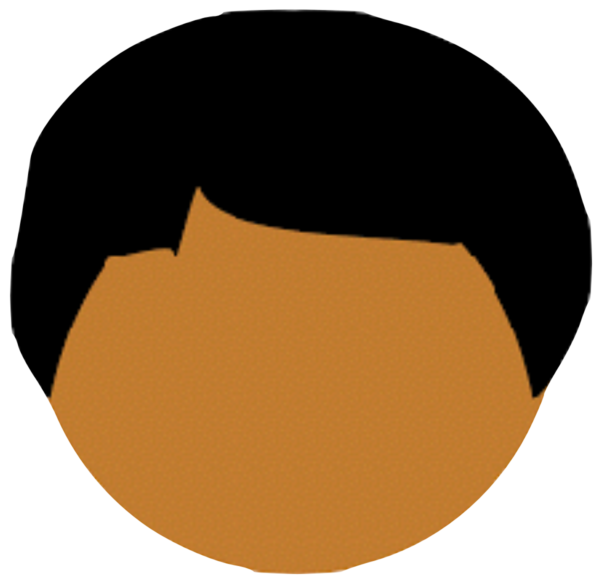


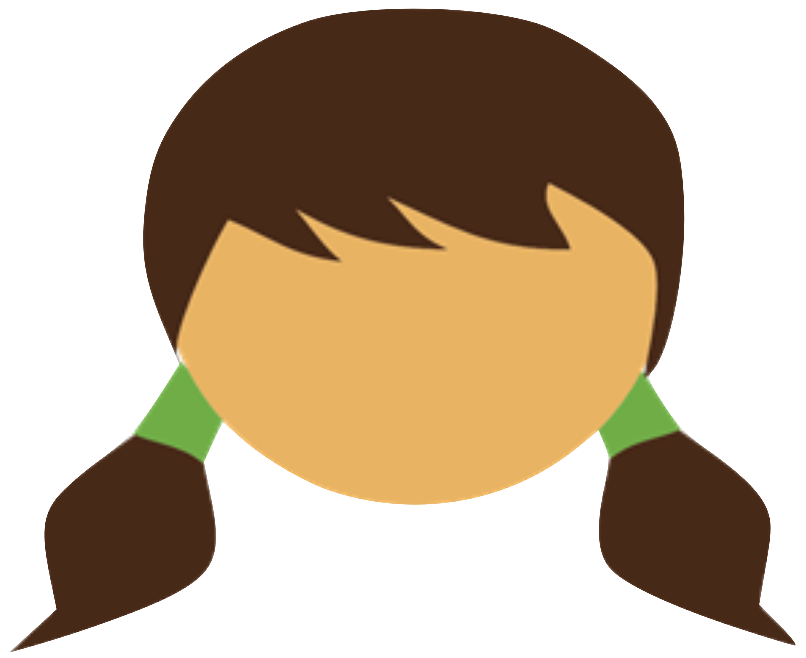


The XOR secret sharing *of a bit x is a pair of bits $\langle x_0, x_1 \rangle$ where P_0 holds x_0 and P_1 holds x_1, and where $x_0 \oplus x_1 = x$*

We sometimes denote such a pair by $[x]$

XOR Secret Shares

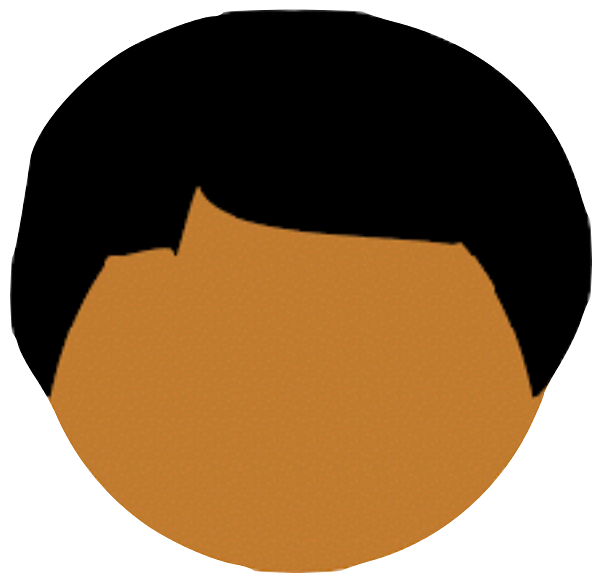




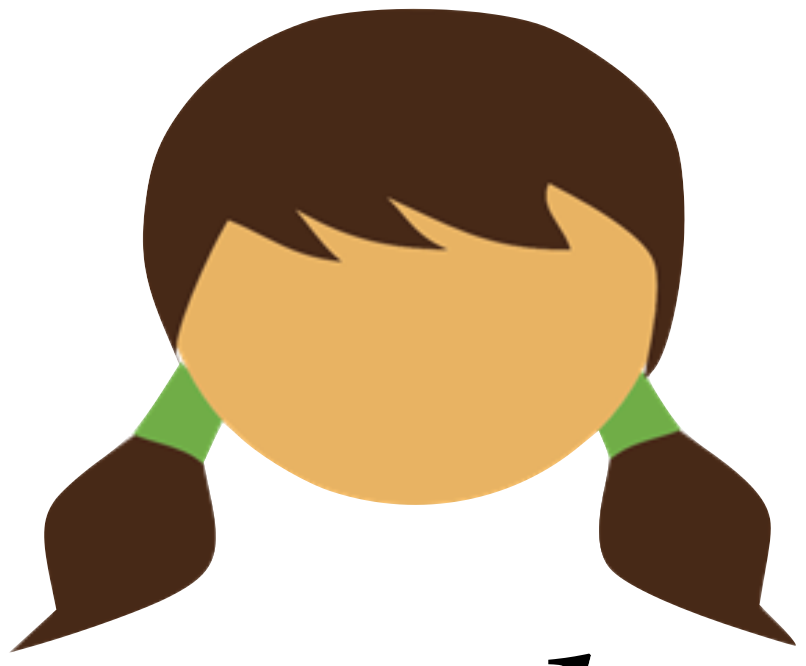
The XOR secret sharing *of a bit x is a pair of bits $\langle x_0, x_1 \rangle$ where P_0 holds x_0 and P_1 holds x_1, and where $x_0 \oplus x_1 = x$*

We sometimes denote such a pair by $[x]$

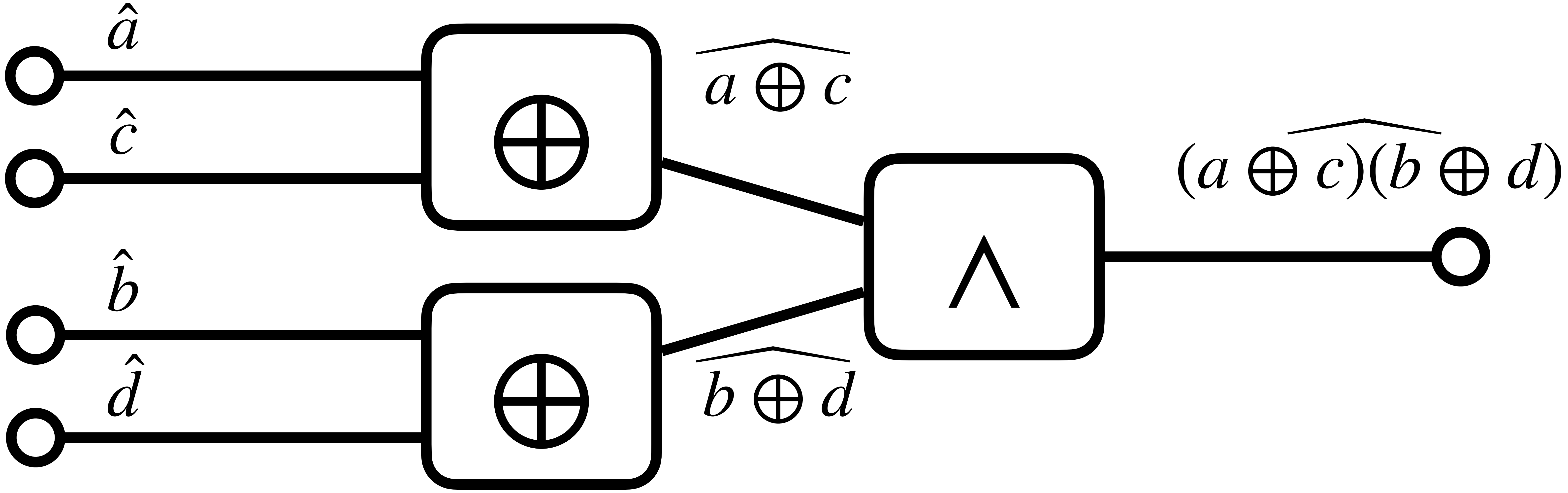
Intuition: P_0's share x_0 acts as a mask, hiding x from P_1 (and vice versa)

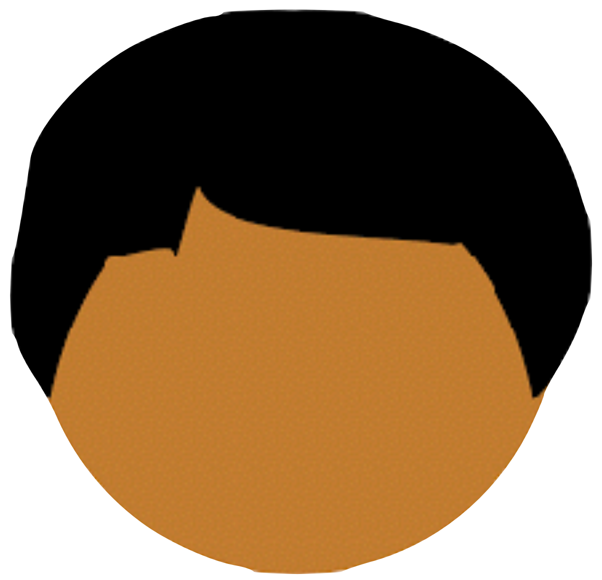


a, b

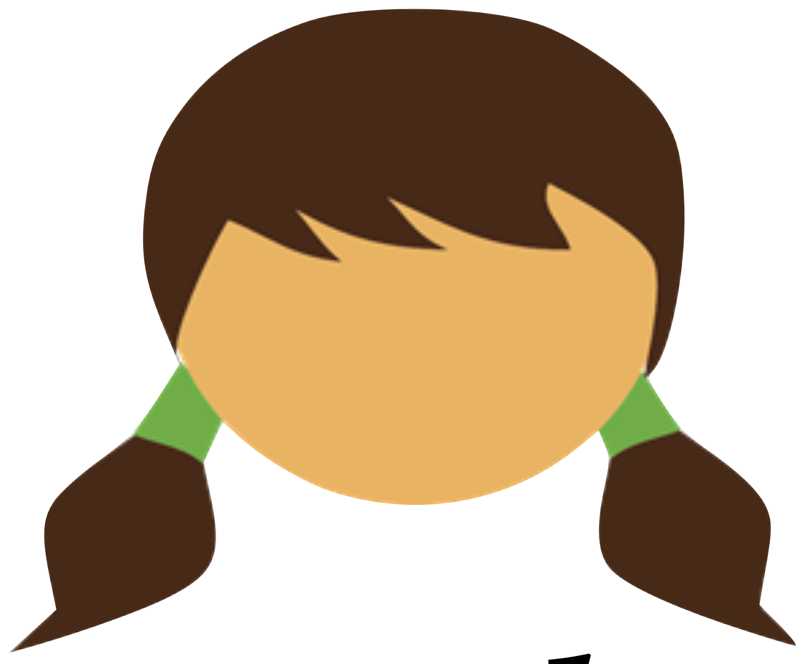


c, d

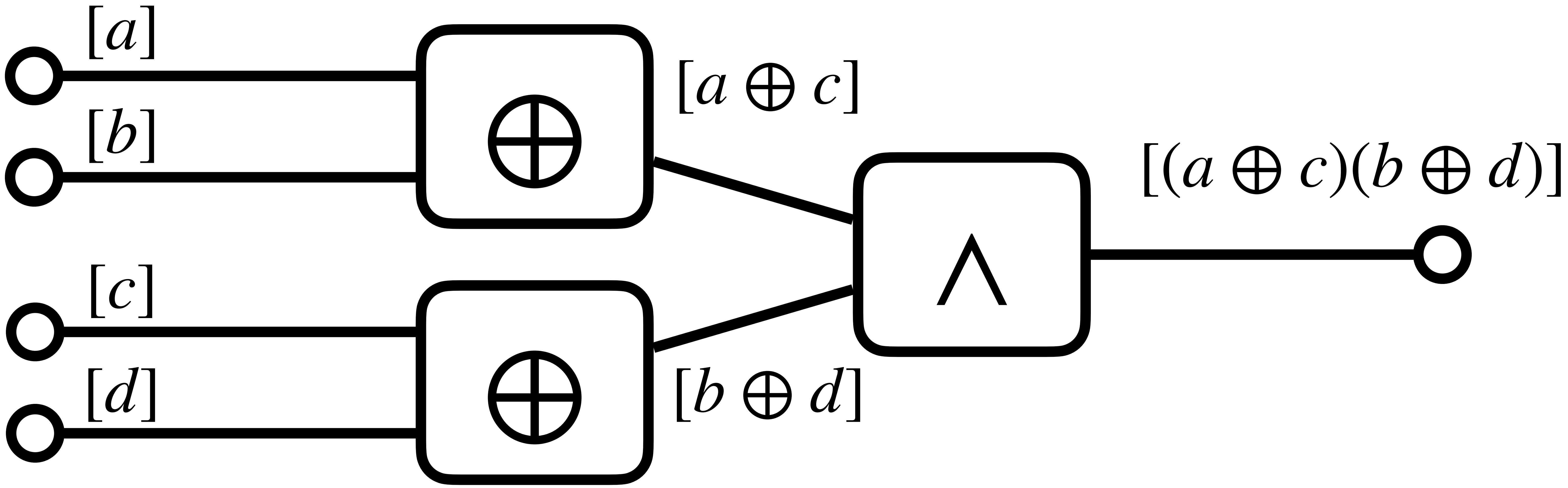

$\hat{a}$
$\hat{c}$
$\hat{b}$
$\hat{d}$
$\oplus$
$\oplus$
$\widehat{a \oplus c}$
$\widehat{b \oplus d}$
$\wedge$
$\widehat{(a \oplus c)(b \oplus d)}$

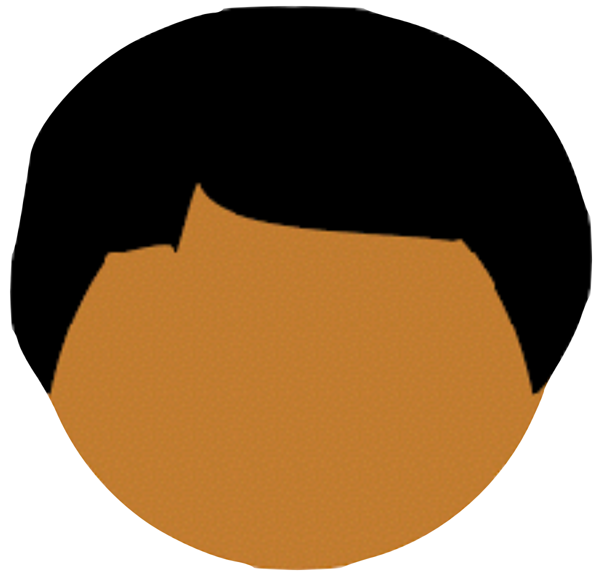


a, b



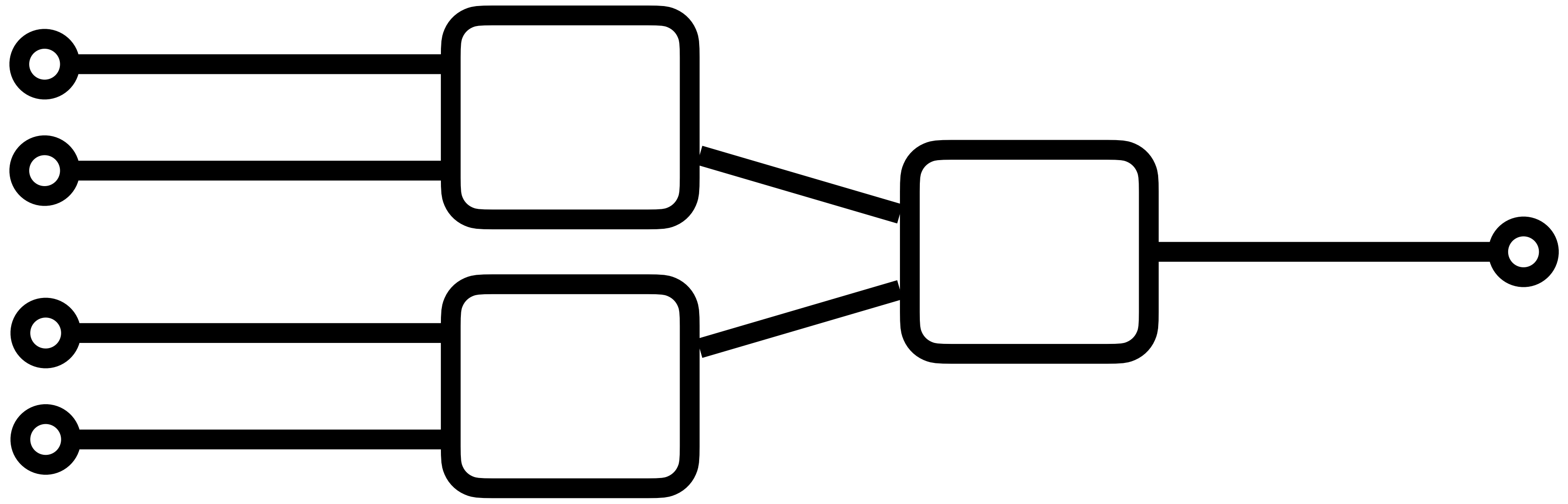
c, d

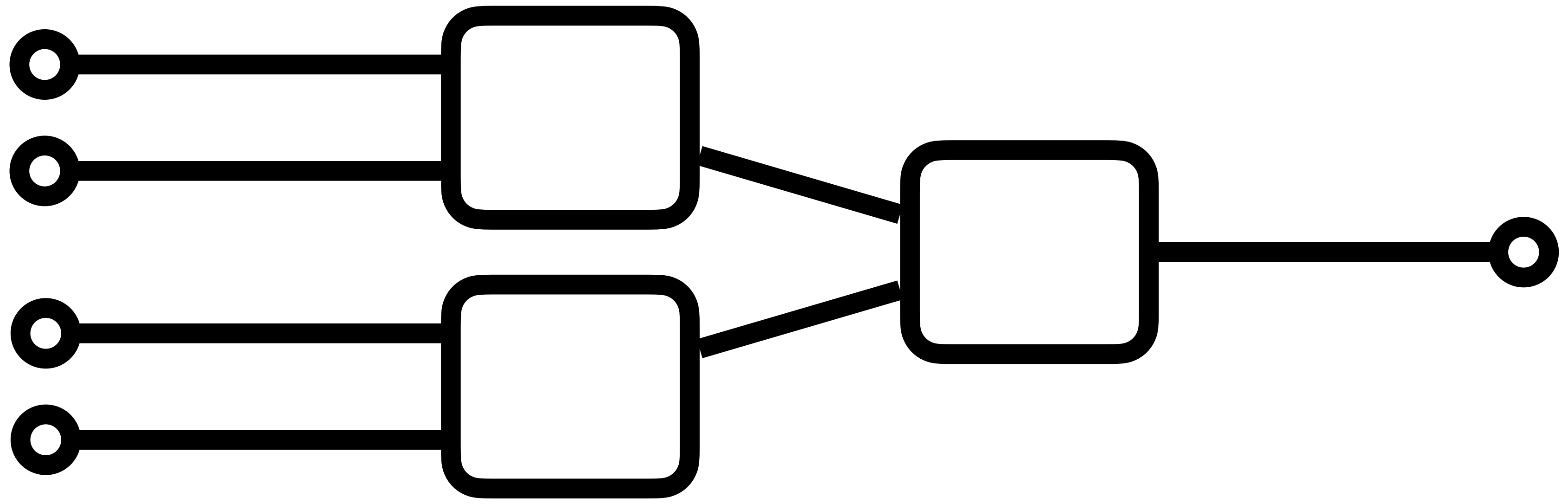




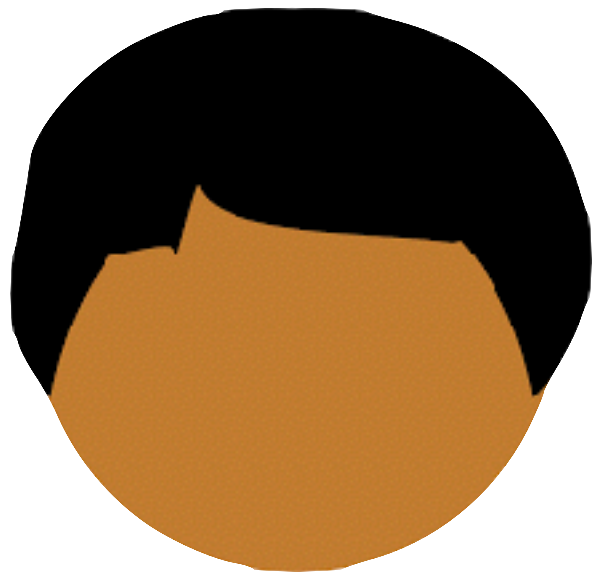
a, b

c, d





Each party in its head maintains a local copy of the circuit, placing its shares on the wires



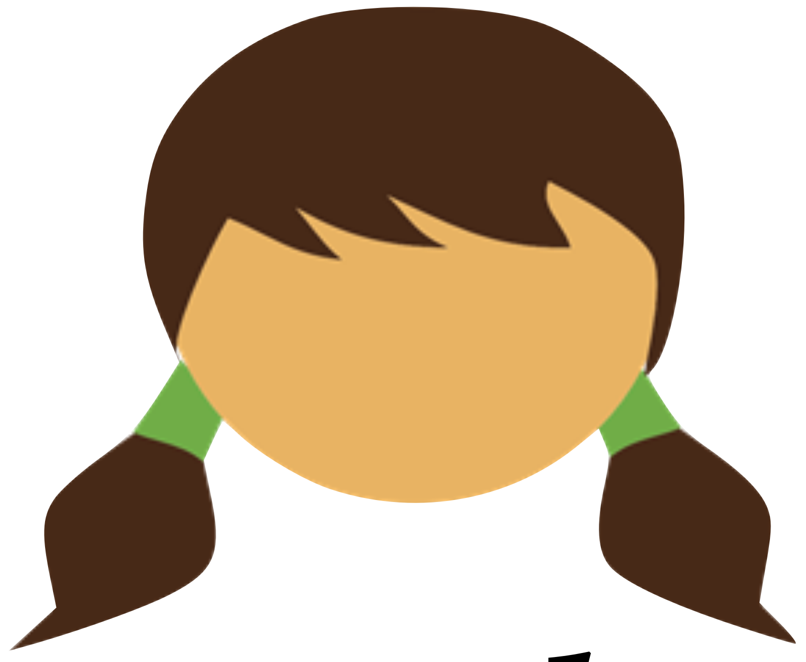
a, b

Where do input shares come from?

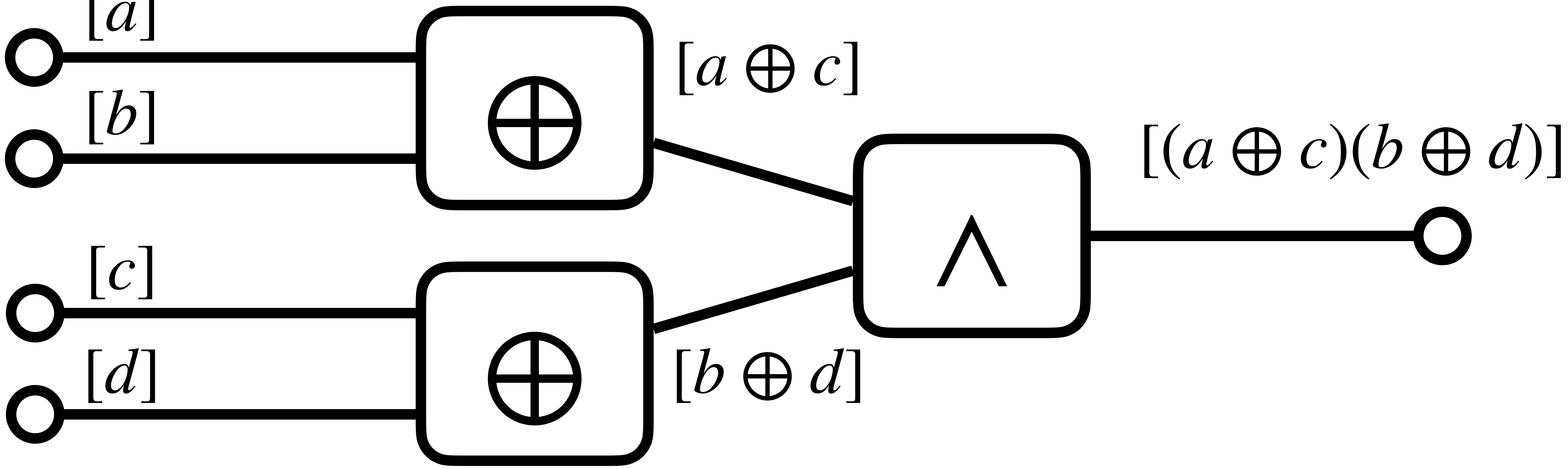
How do we XOR two shares?

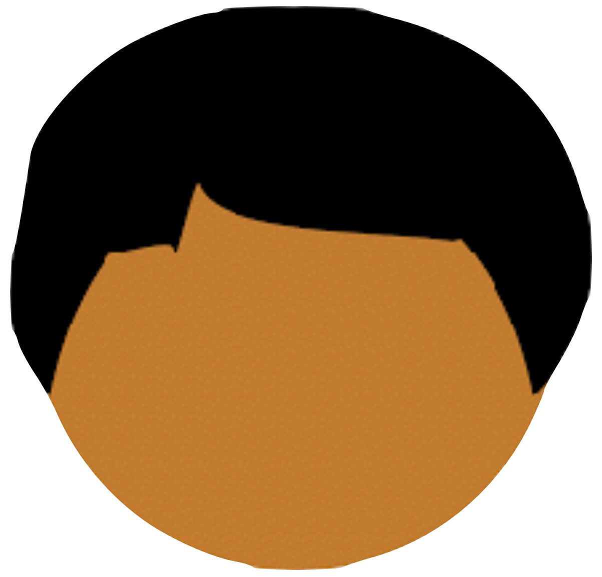
How do we AND two shares?

How do we “decrypt” output shares?



c, d

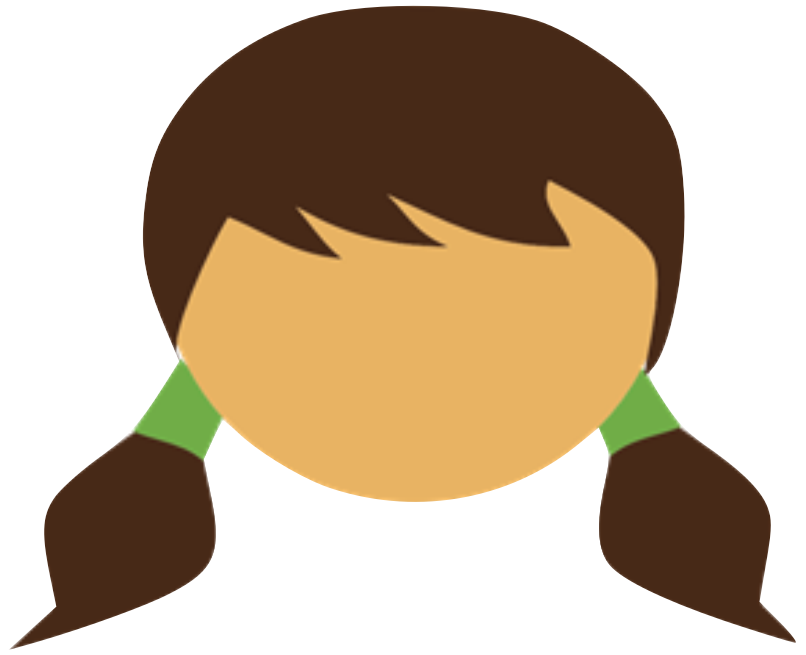




x

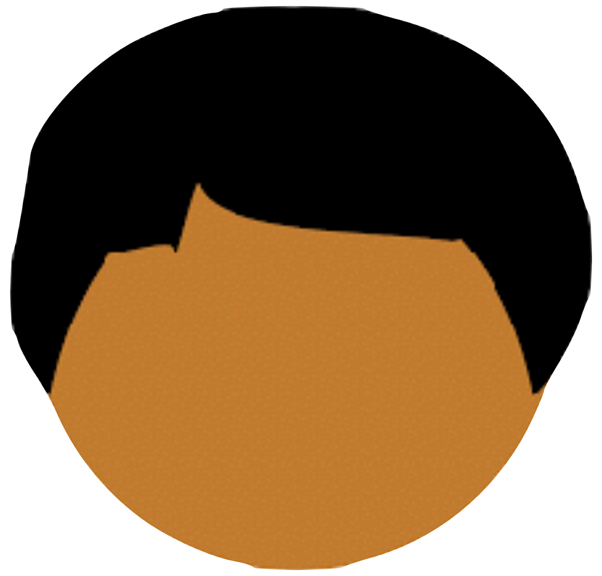
Where do input shares come from?

Goal: put $[x]$ on the input wire



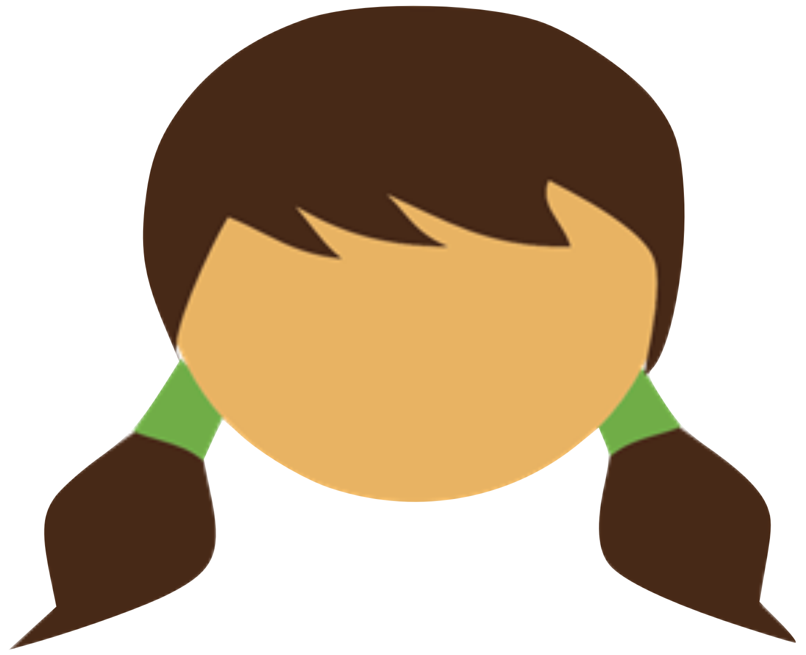
Where do input shares come from?

Goal: put $[x]$ on the input wire



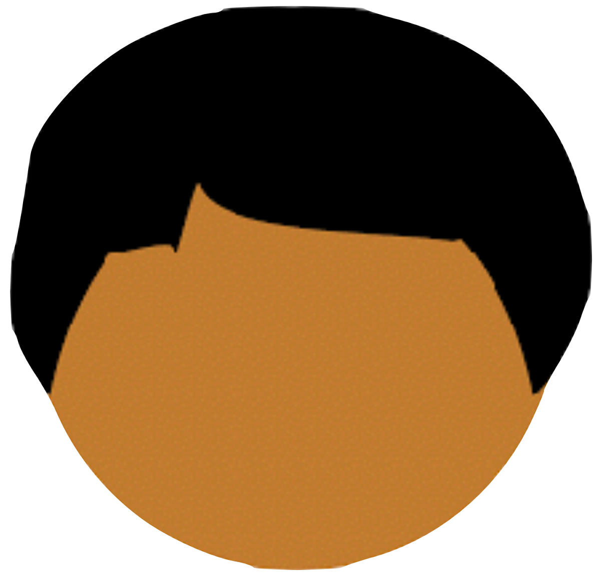
x

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



Where do input shares come from?

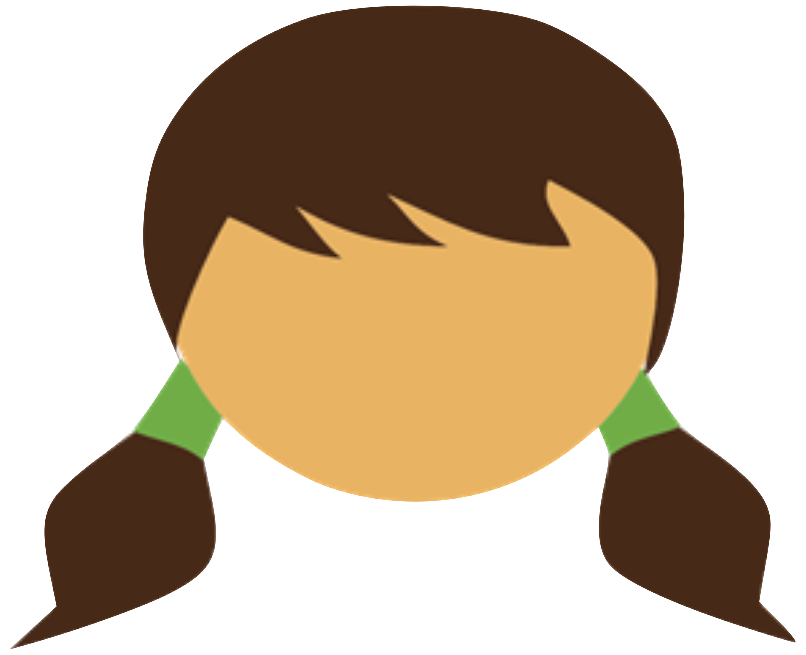
Goal: put $[x]$ on the input wire

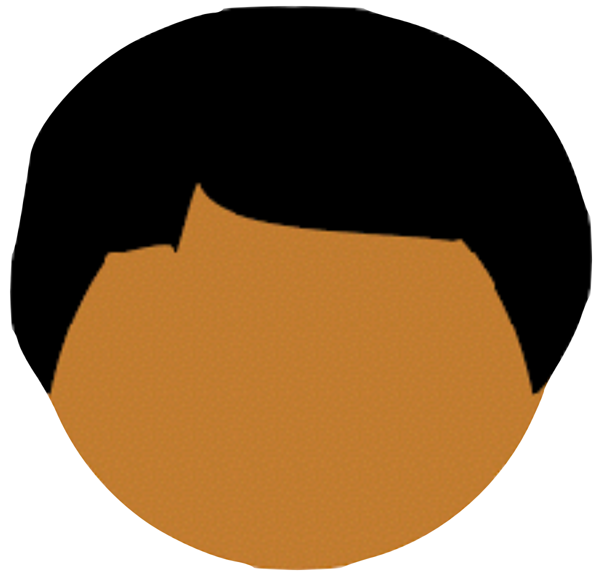


x

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

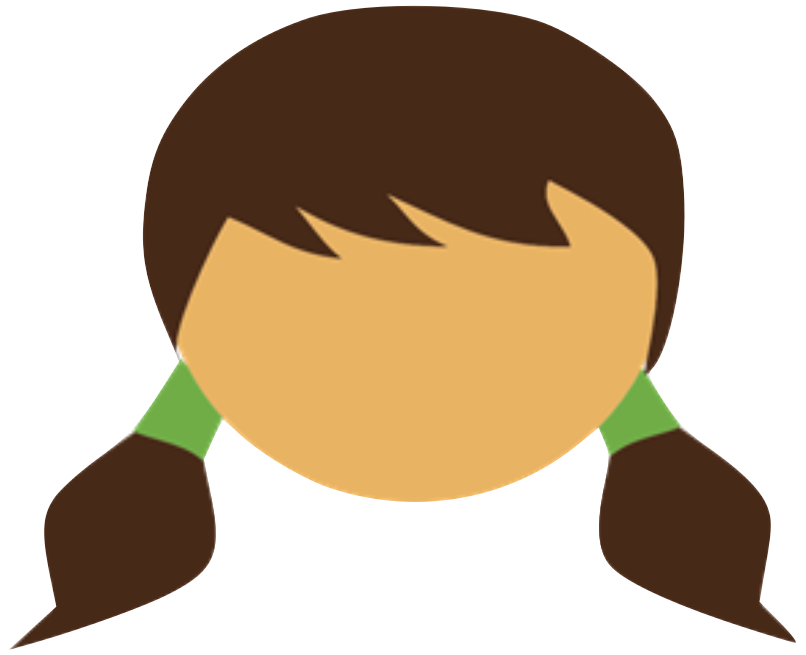
$$\xrightarrow{\quad x \oplus r \quad}$$





Where do input shares come from?

Goal: put $[x]$ on the input wire



x

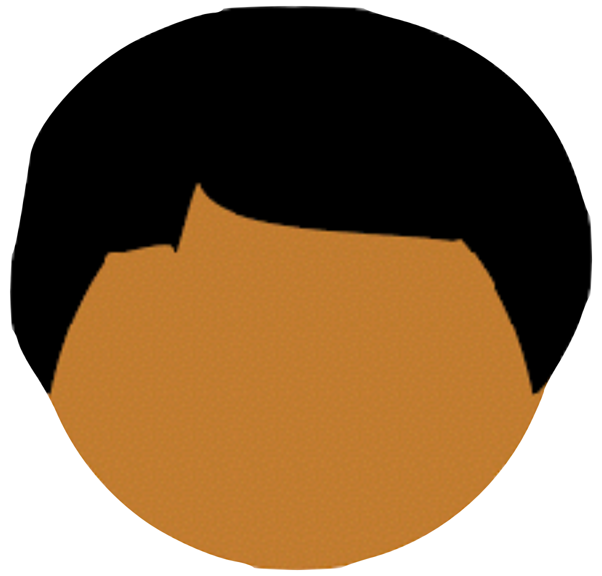
$$r \xleftarrow{\$} \{0,1\}$$

$$x \oplus r$$

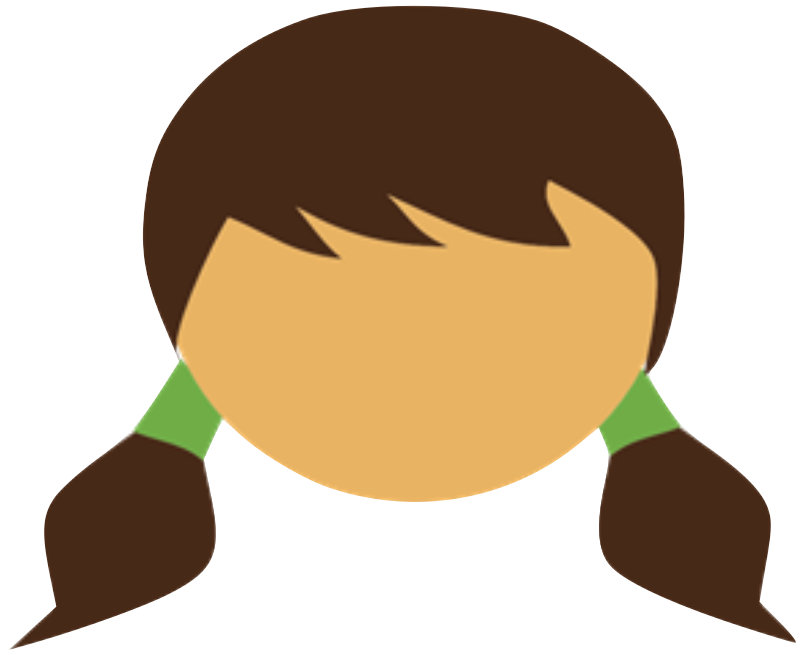
$$\longrightarrow$$

r

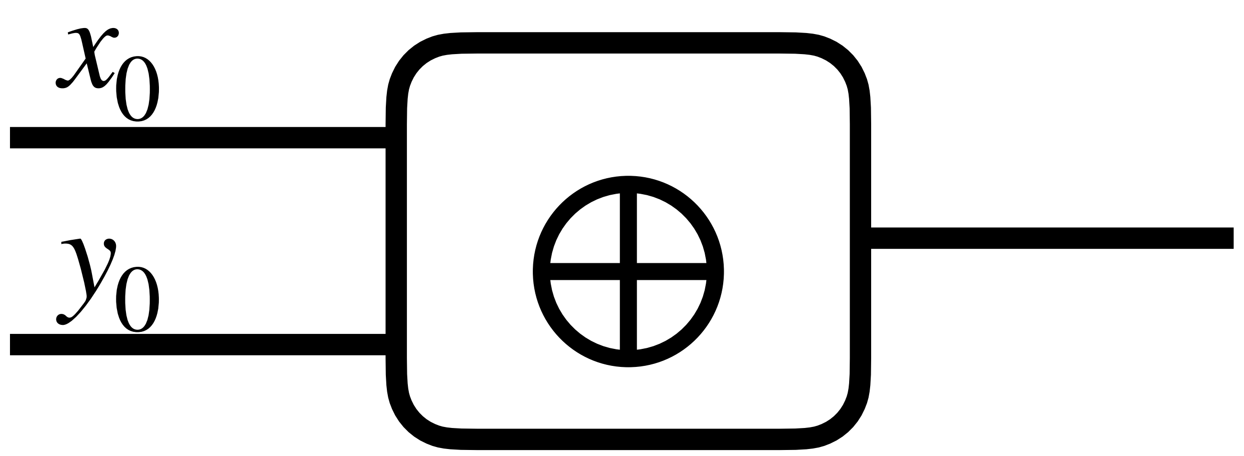
$x \oplus r$

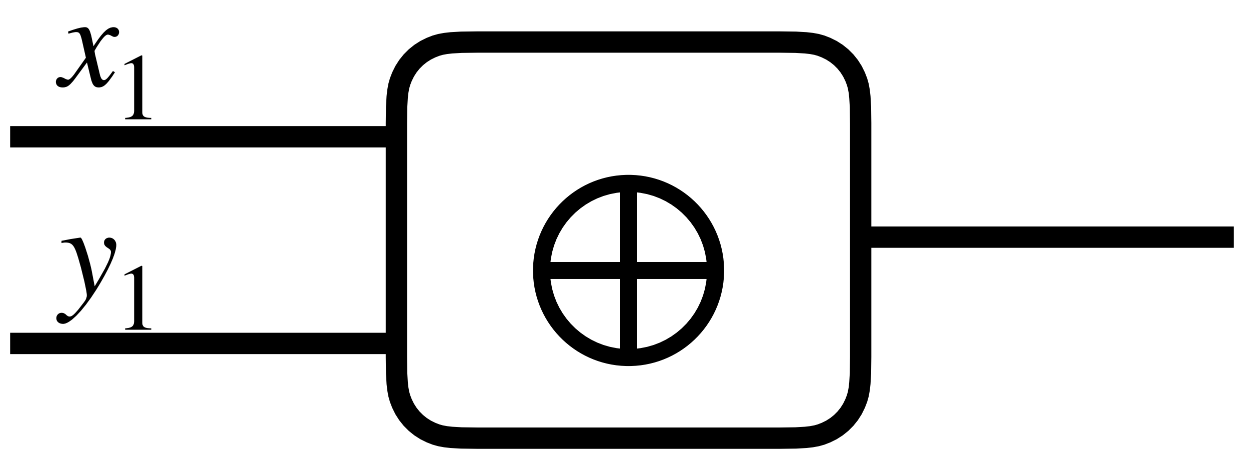


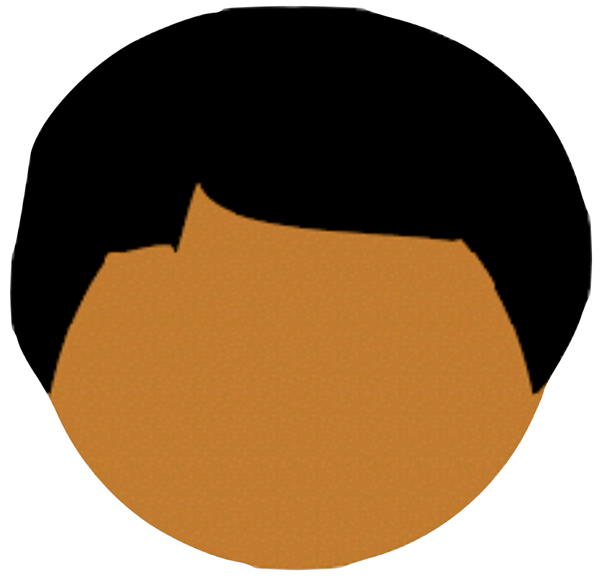
How do we XOR two shares?



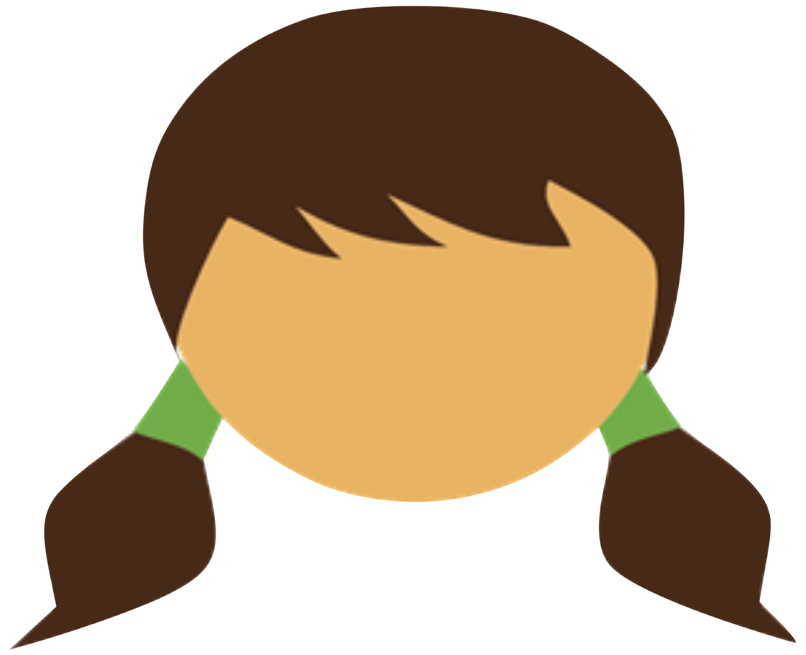
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \oplus y]$ on the gate output



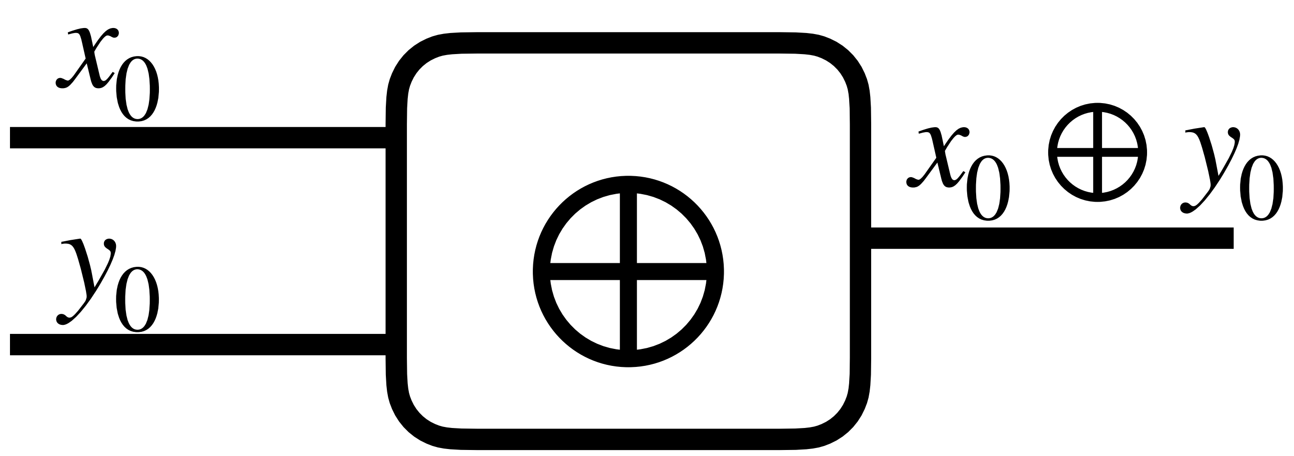


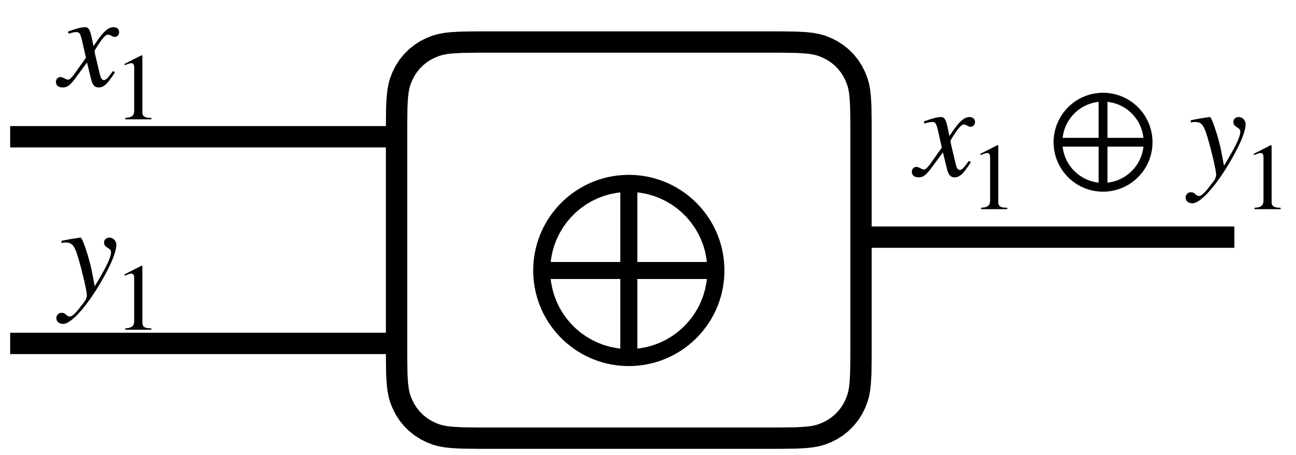


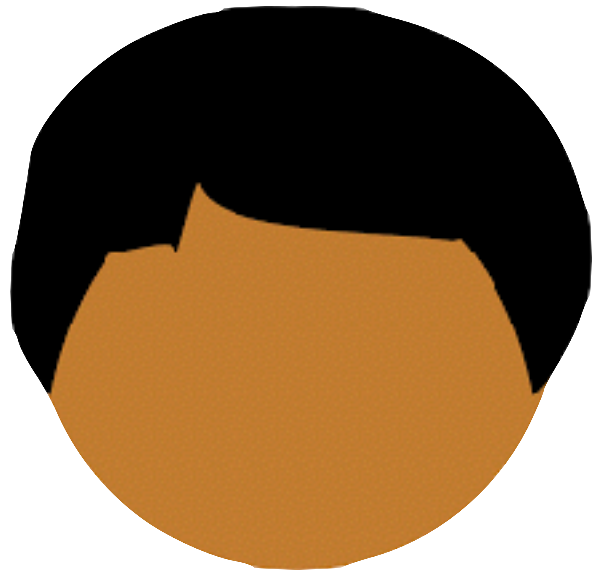
How do we XOR two shares?



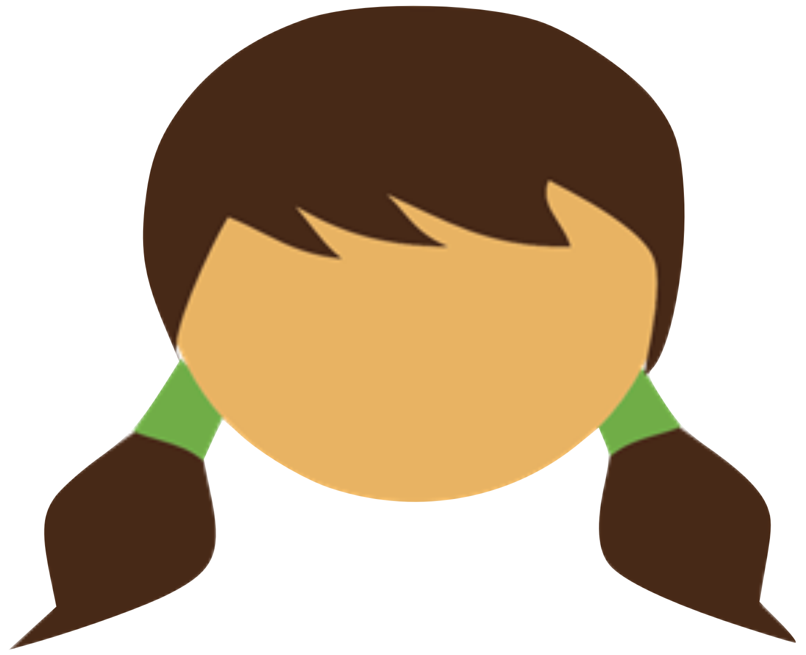
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \oplus y]$ on the gate output



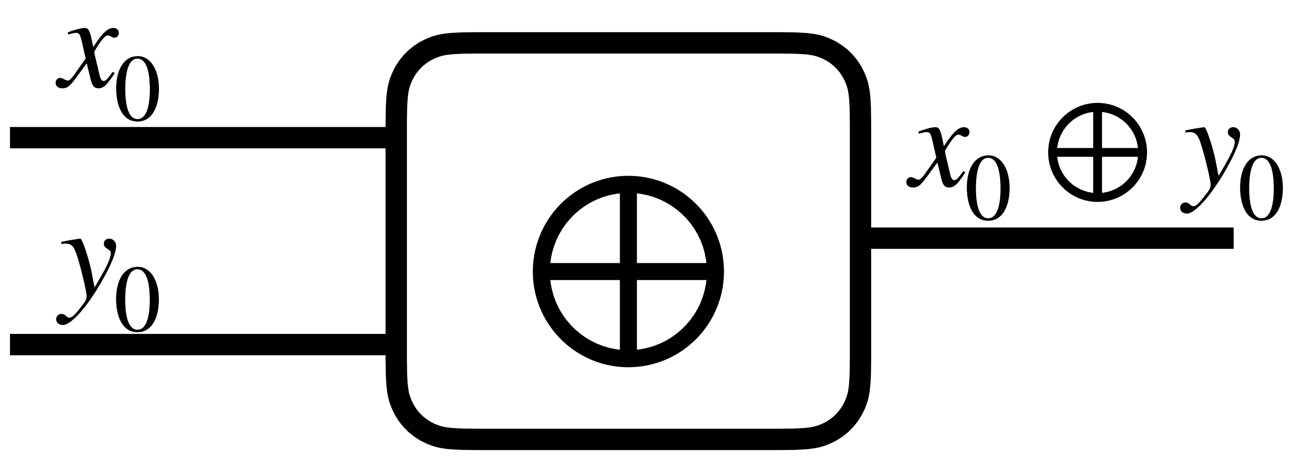




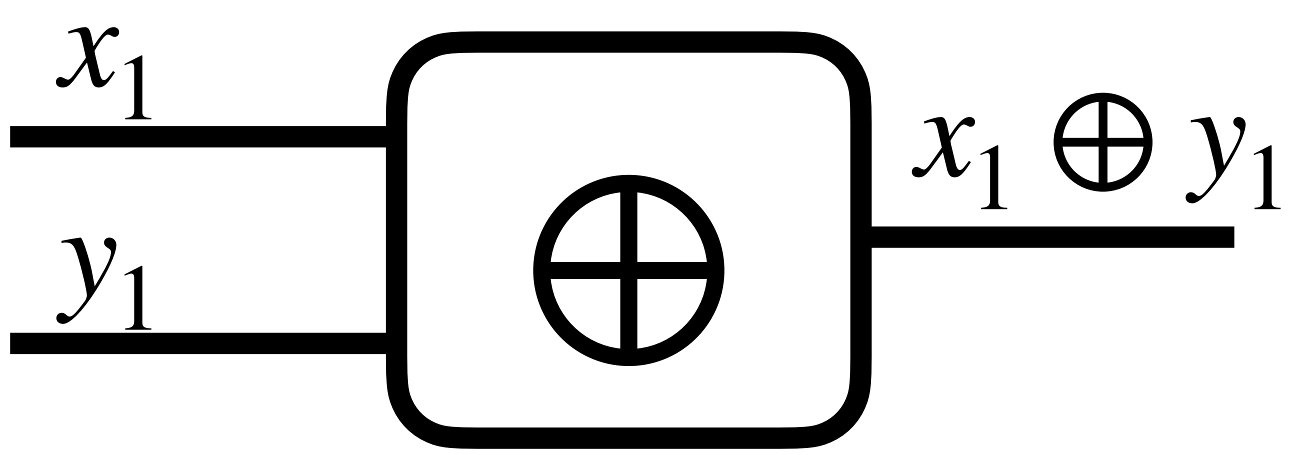
How do we XOR two shares?

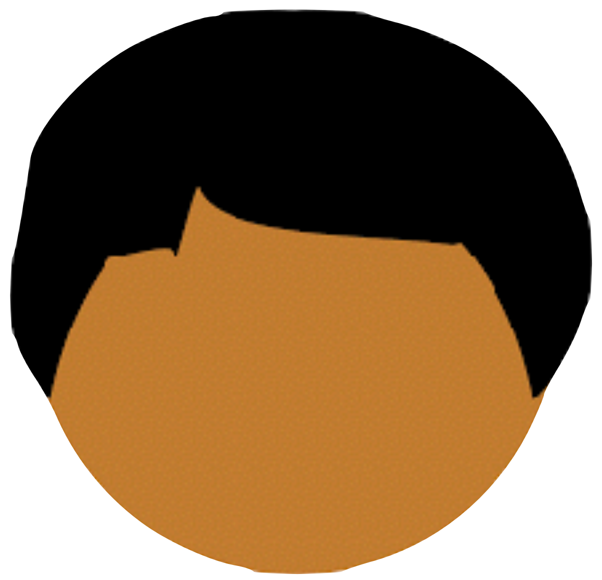


Goal: given gate input wires holding $[x]$, $[y]$, put $[x \oplus y]$ on the gate output

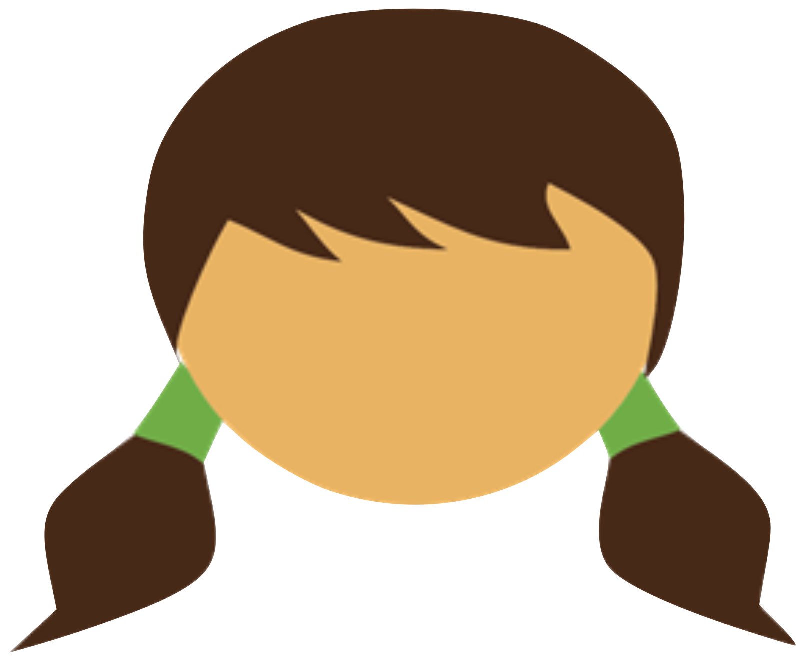


XOR is “free”

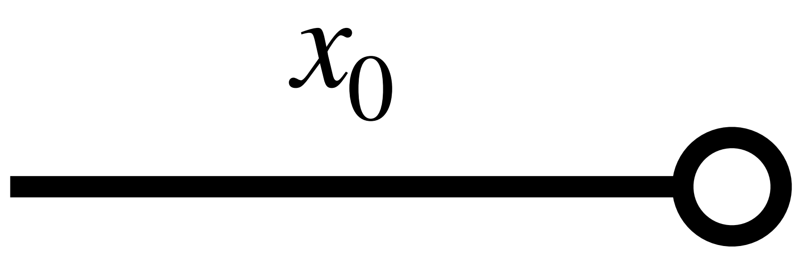


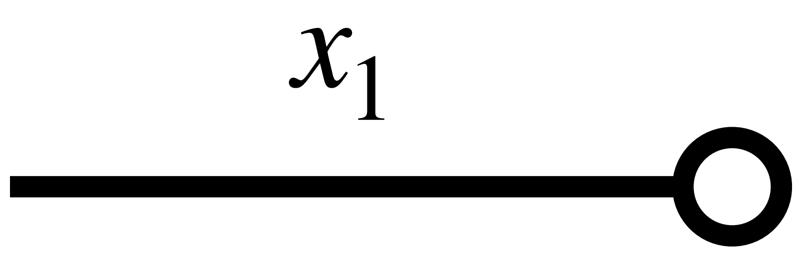


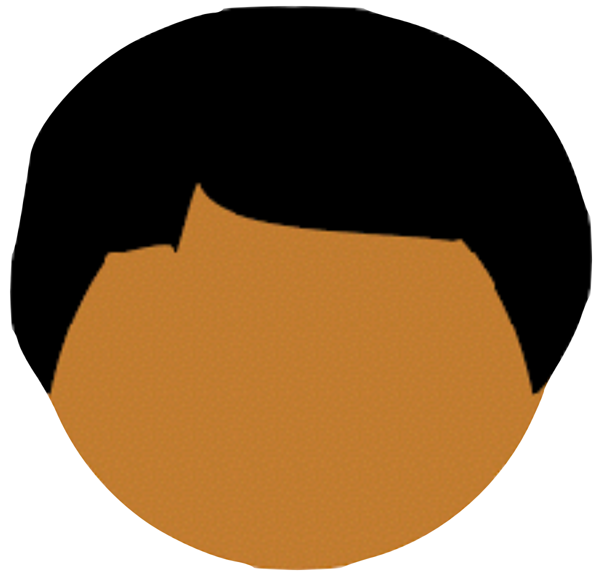
How do we “decrypt” output shares?



Goal: given wire holding $[x]$,
reveal x to each party

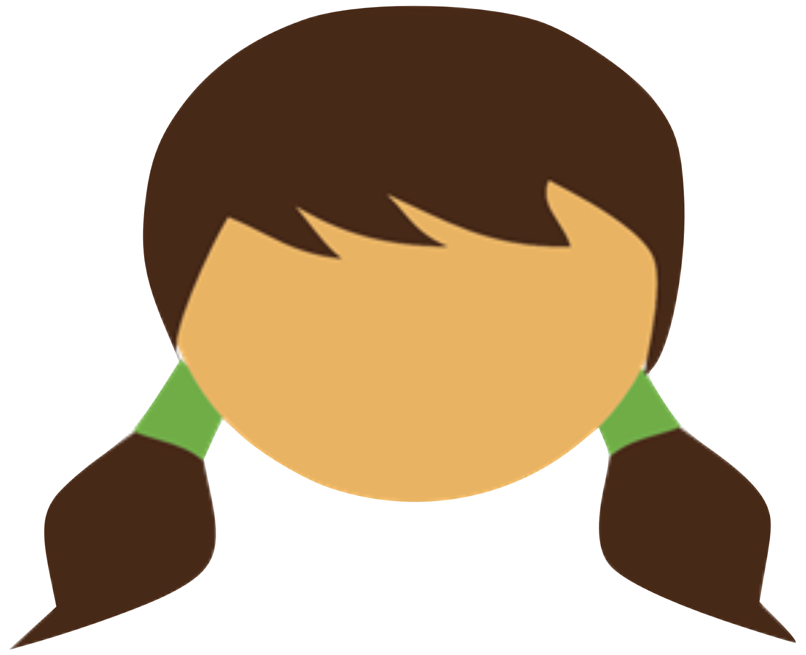


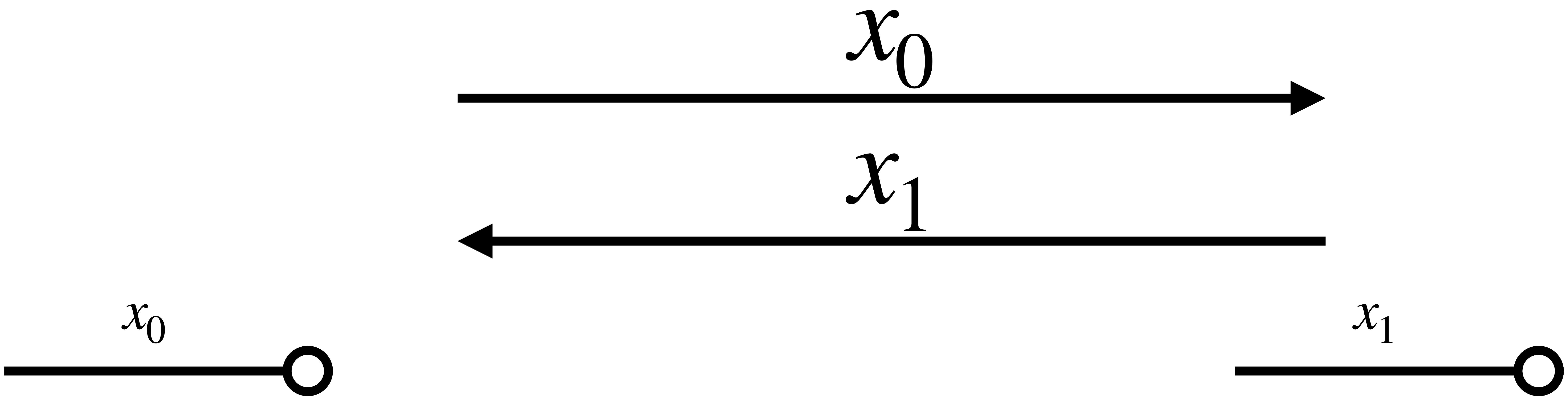


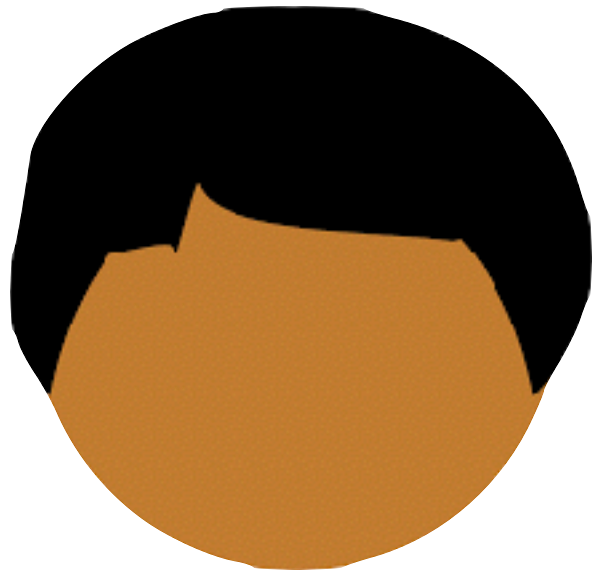


How do we "decrypt" output shares?

Goal: given wire holding $[x]$,
reveal x to each party





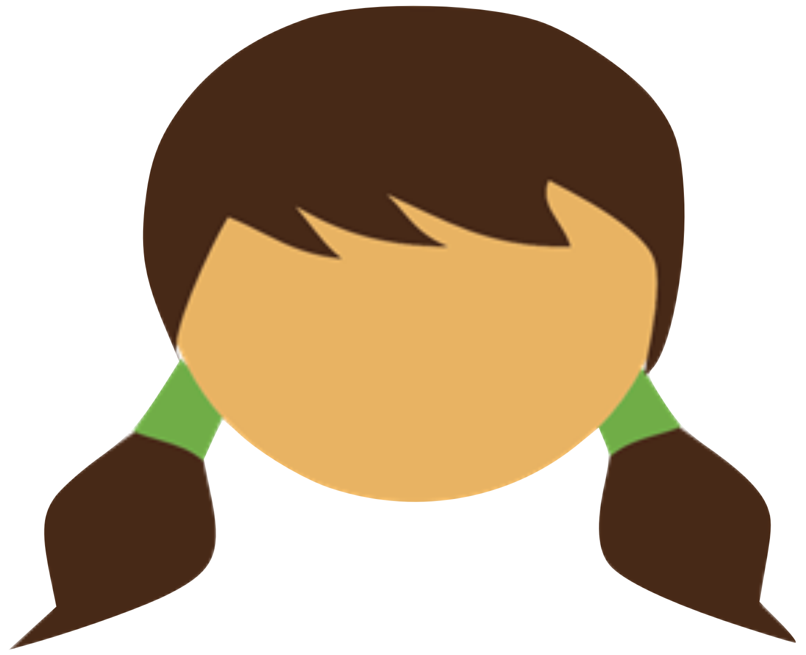


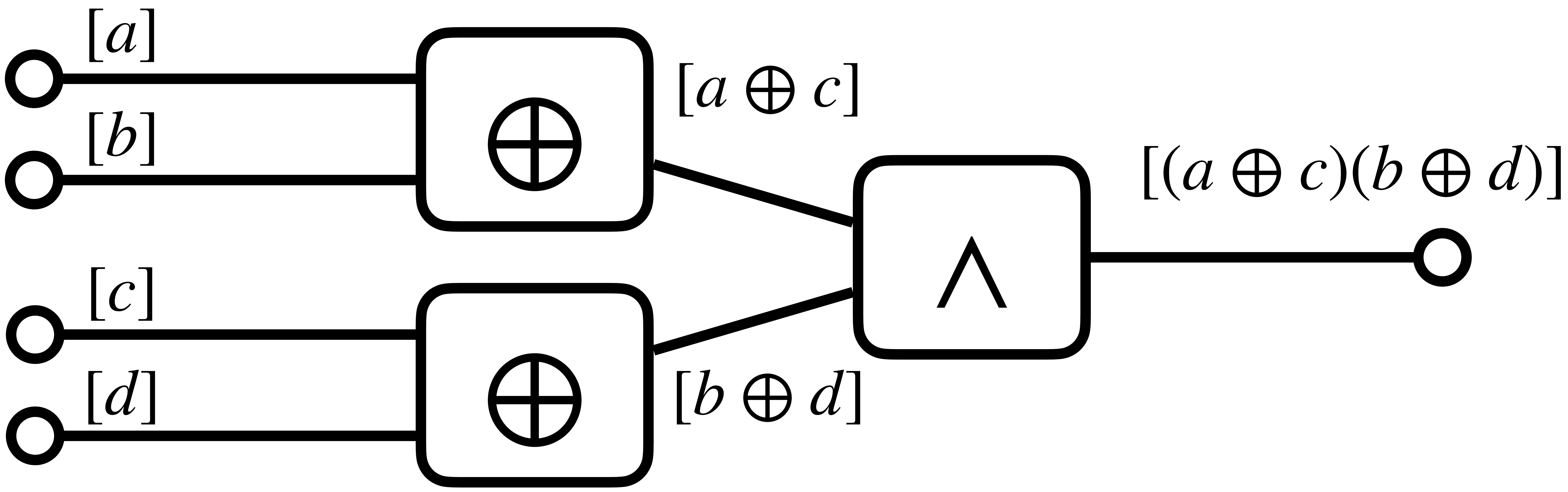
Where do input shares come from?

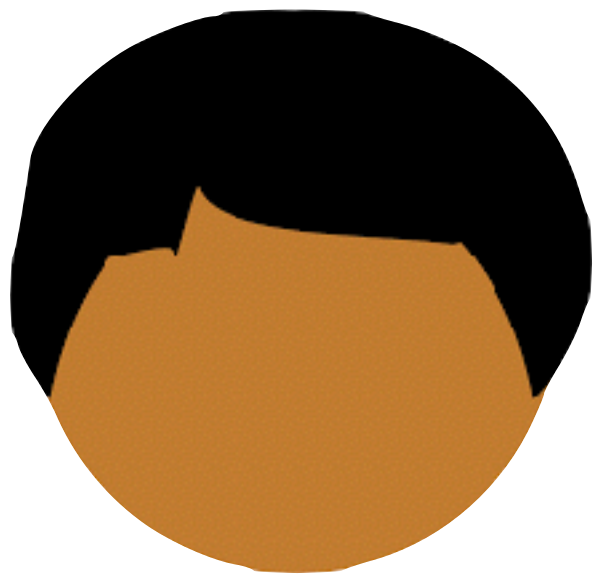
How do we XOR two shares?

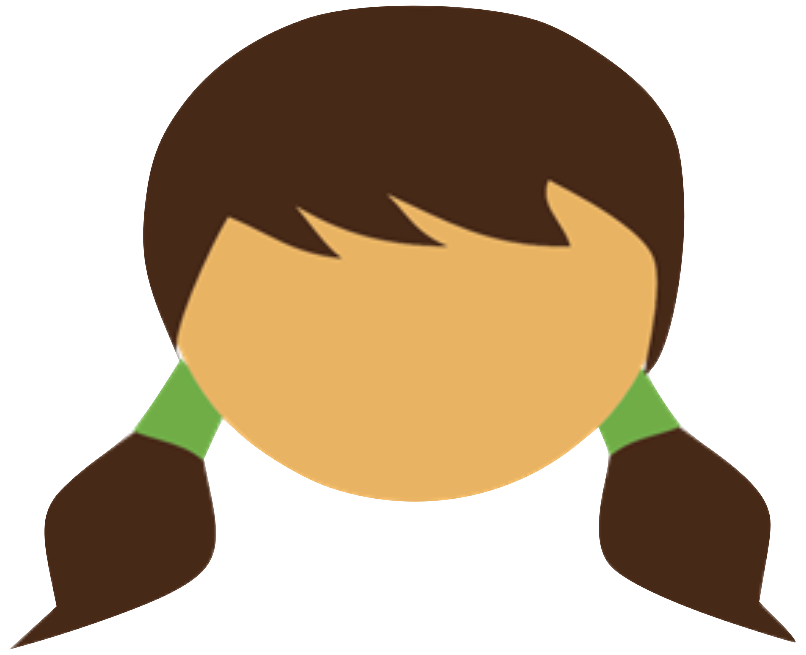
How do we AND two shares?

How do we “decrypt” output shares?



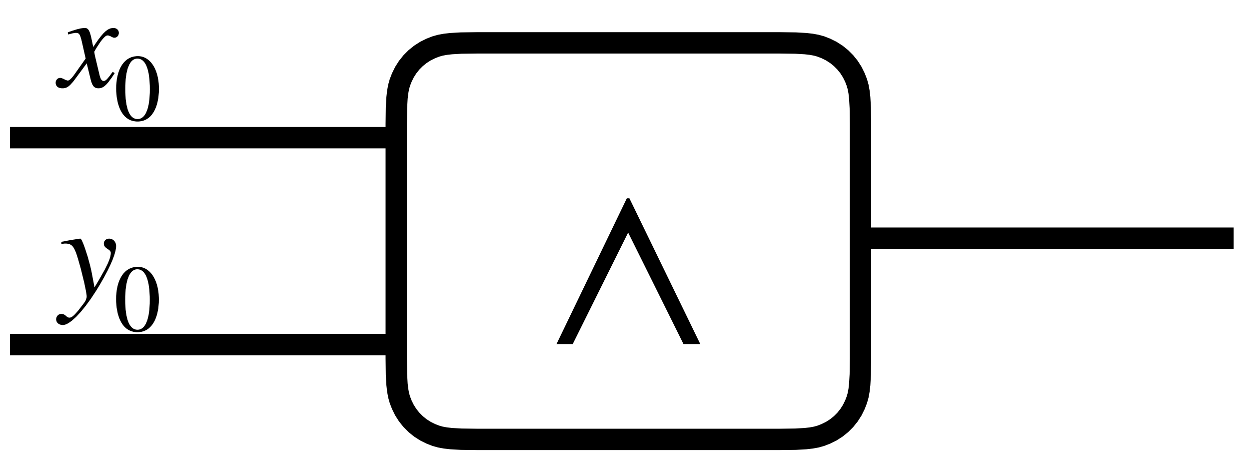


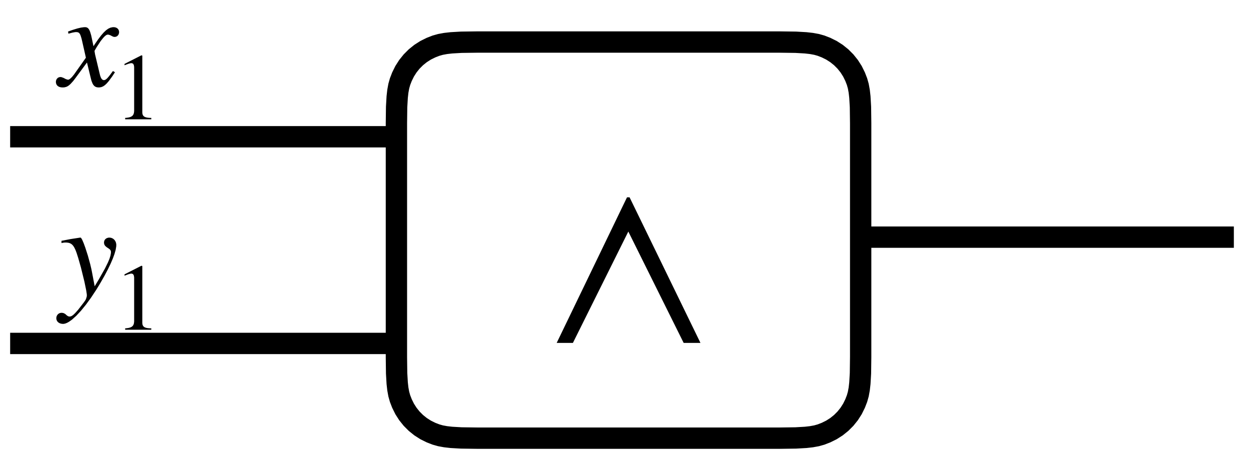


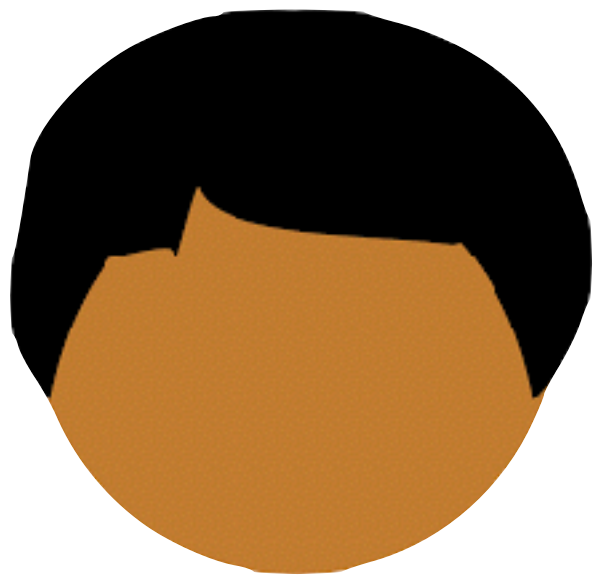


How do we AND two shares?

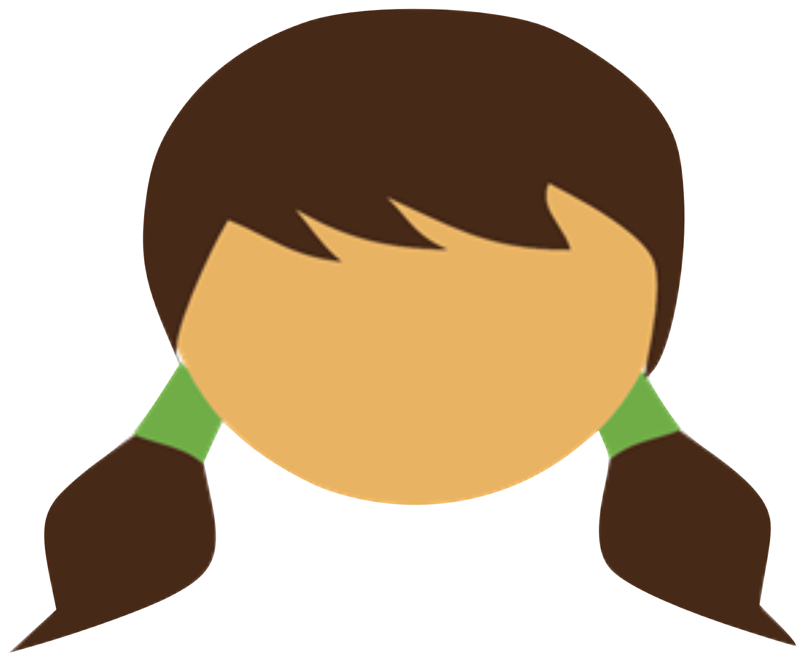
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \wedge y]$ on the gate output





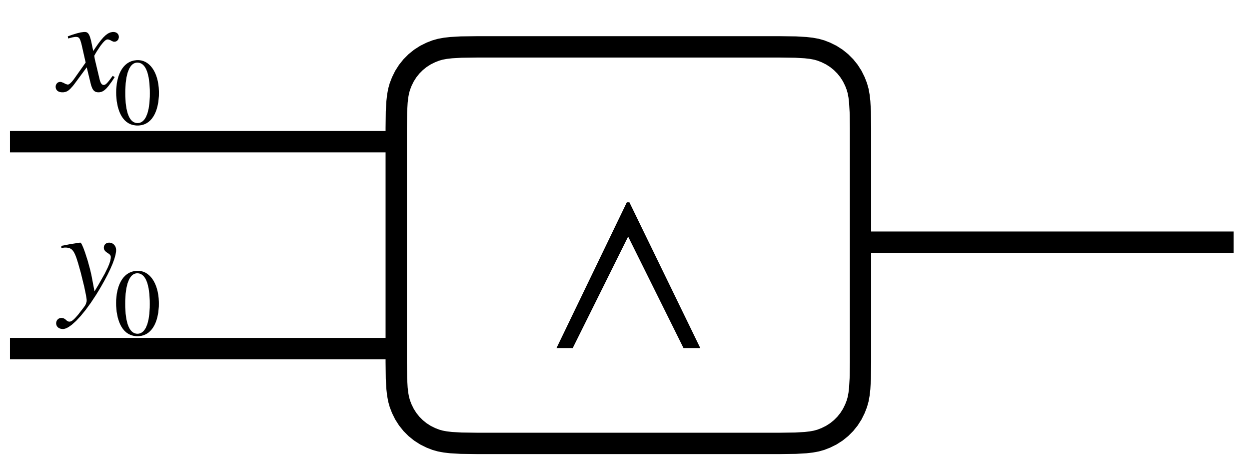


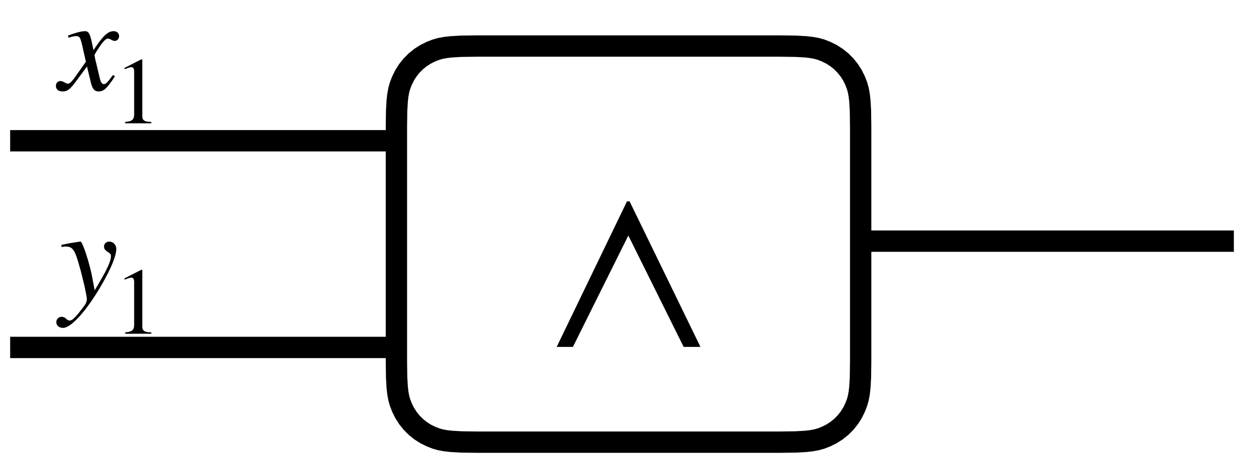
How do we AND two shares?

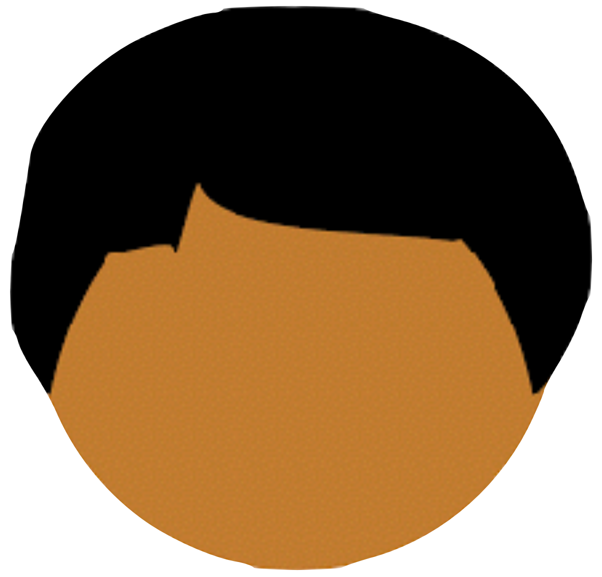


Goal: given gate input wires holding $[x]$, $[y]$, put $[x \wedge y]$ on the gate output

$$(x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$$





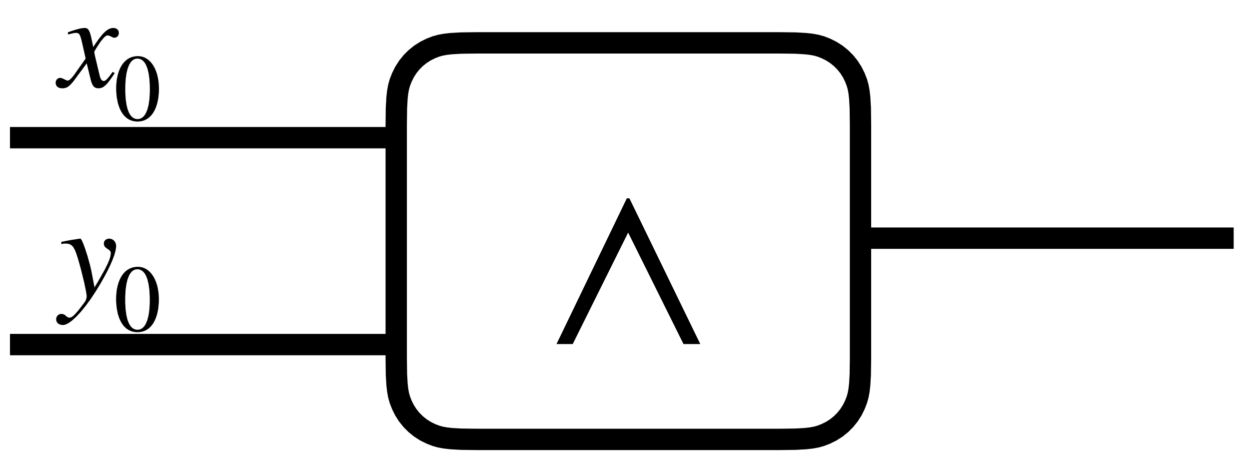


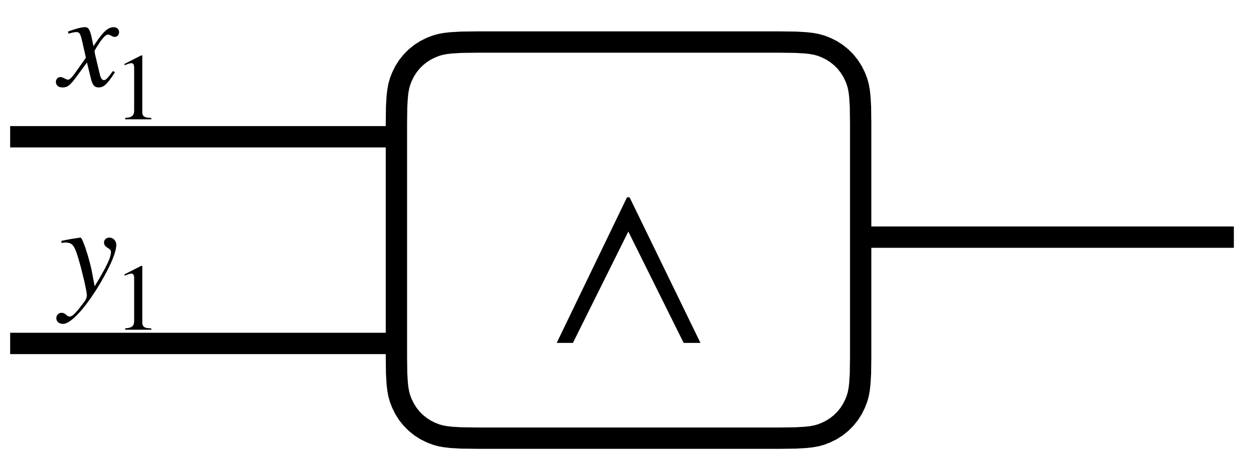
How do we AND two shares?

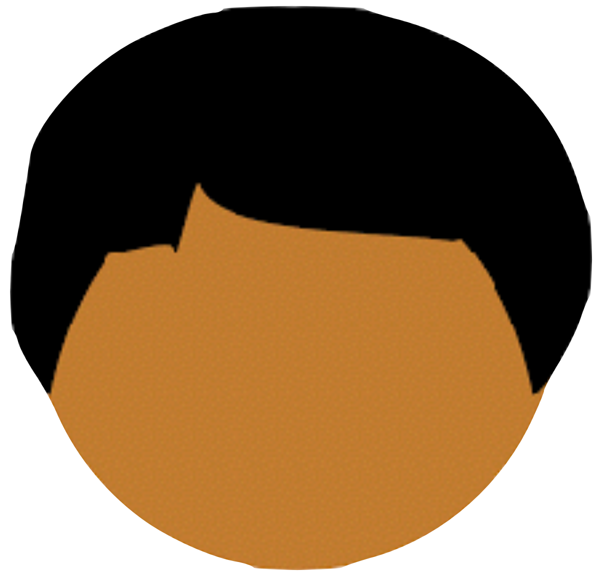
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \wedge y]$ on the gate output

$$(x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$$

$$= (x_0 \wedge y_0) \oplus (x_0 \wedge y_1) \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1)$$

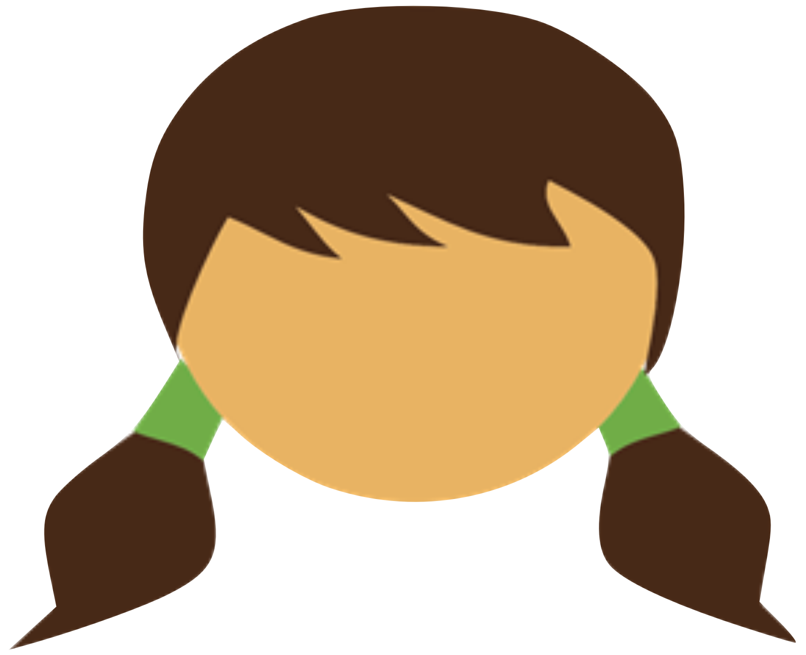






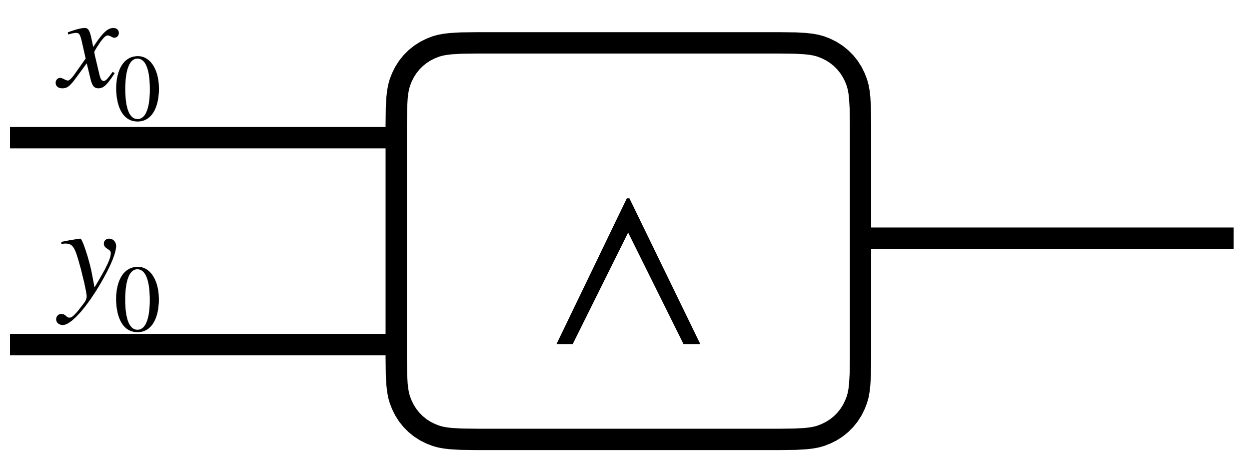
How do we AND two shares?

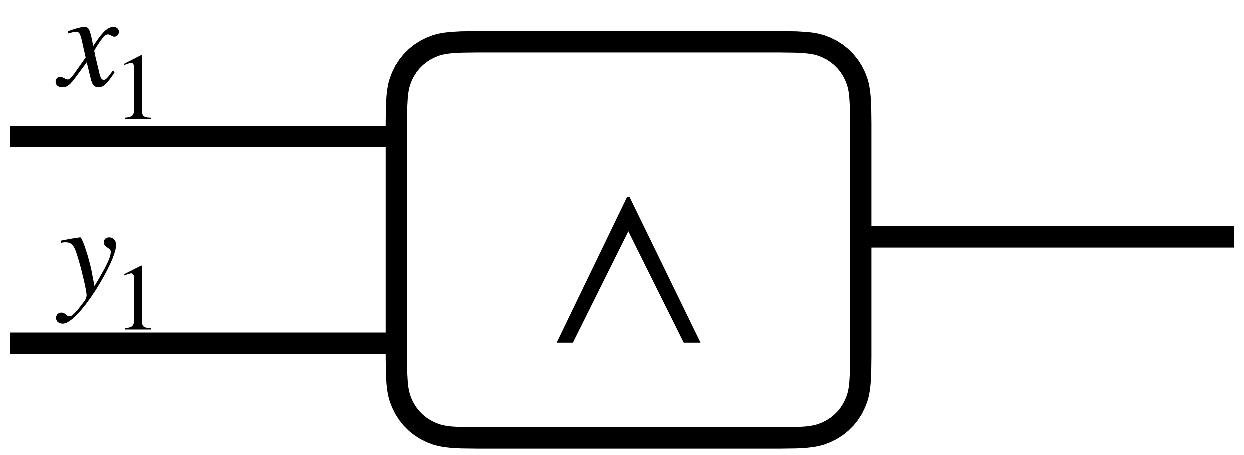
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \wedge y]$ on the gate output

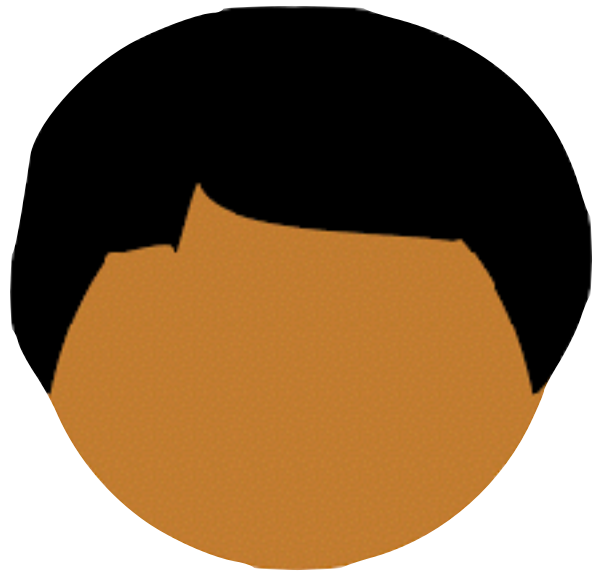


$$(x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$$
$$= (x_0 \wedge y_0) \oplus (x_0 \wedge y_1) \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1)$$

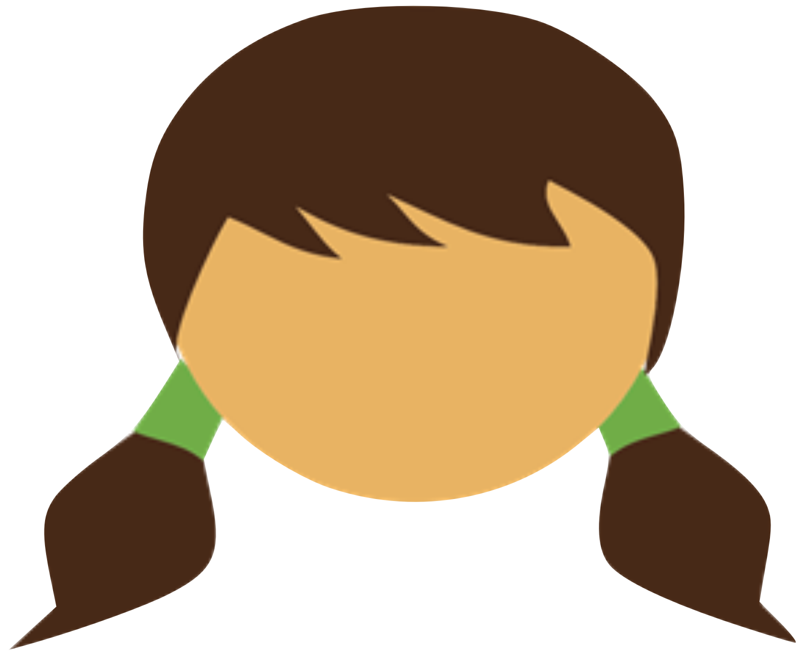
“Free”







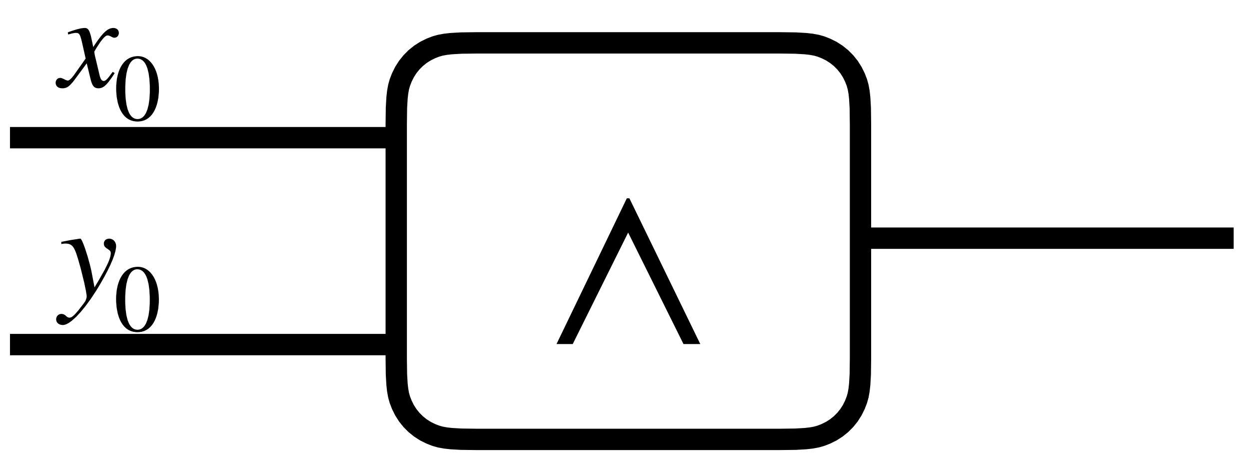
How do we AND two shares?

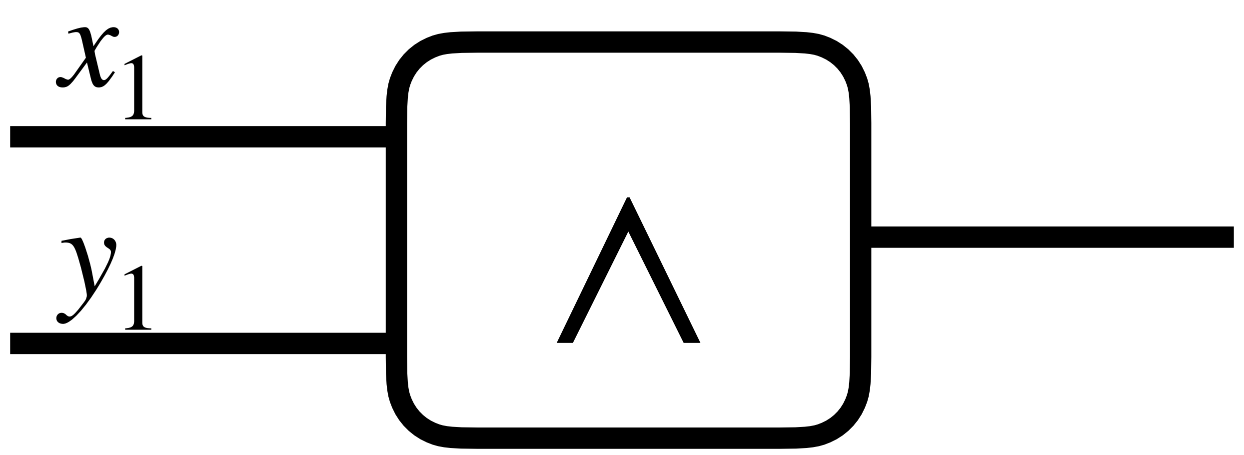


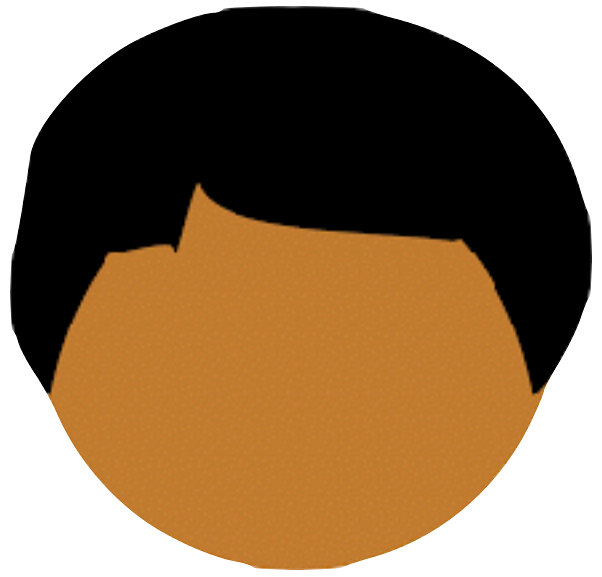
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \wedge y]$ on the gate output

$$(x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$$
$$= (x_0 \wedge y_0) \oplus (x_0 \wedge y_1) \oplus (x_1 \wedge y_0) \oplus (x_1 \wedge y_1)$$

OT



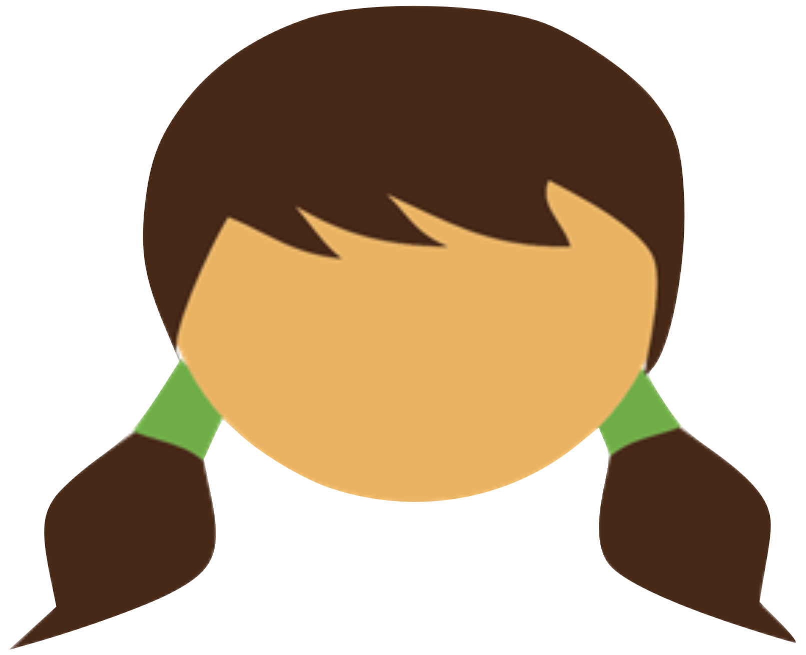




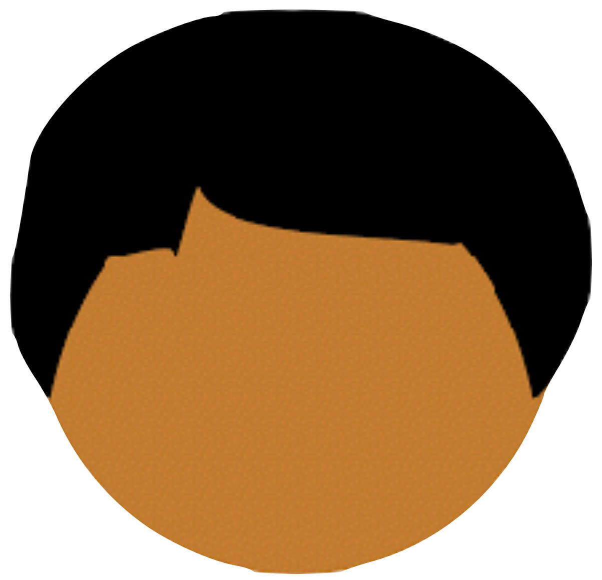
x

Important Subgoal

Goal: given gate input bits x, y, compute random secret share $[x \wedge y]$ s.t. neither party learns $x \wedge y$



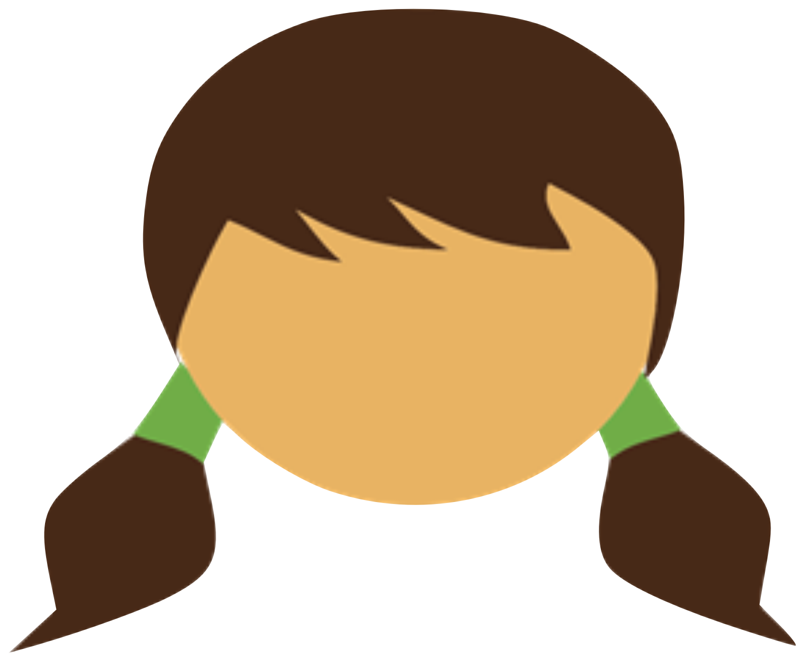
y



x

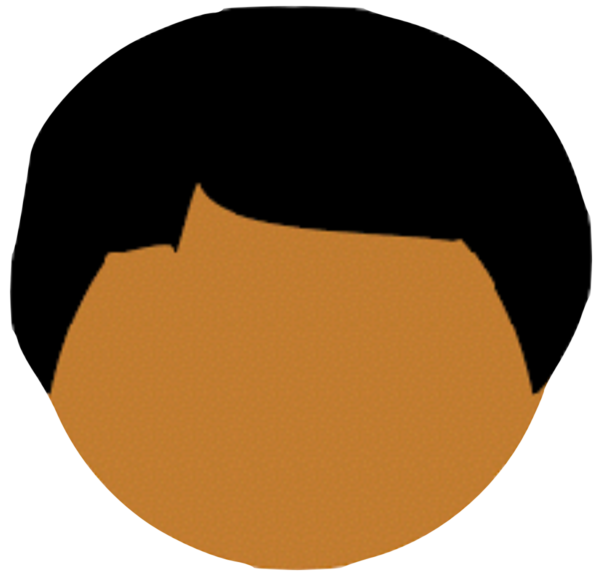
Important Subgoal

Goal: given gate input bits x, y, compute random secret share $[x \wedge y]$ s.t. neither party learns $x \wedge y$



y

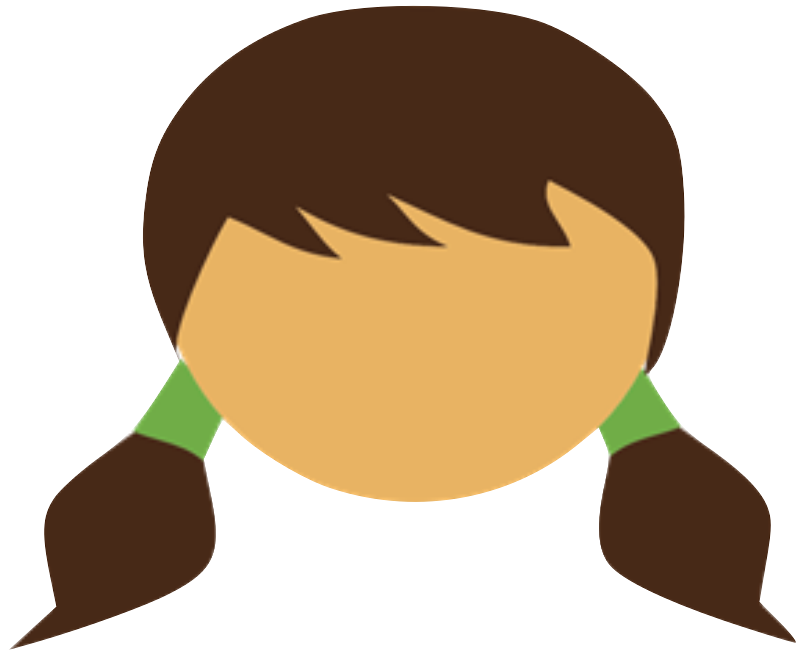
$$r \xleftarrow{\$} \{0,1\}$$



x

Important Subgoal

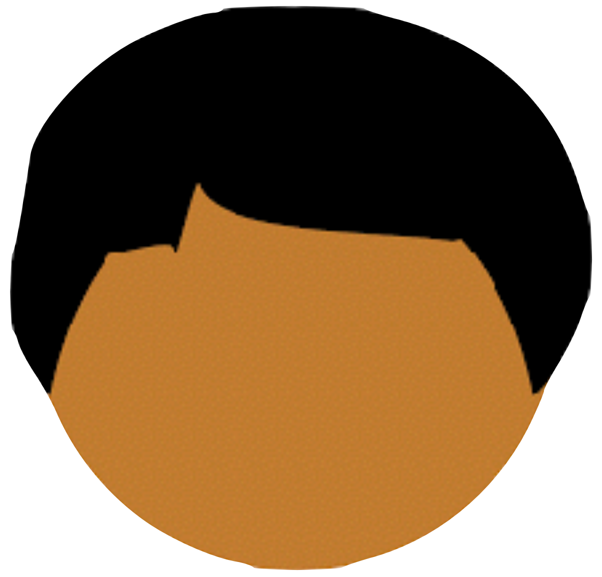
Goal: given gate input bits x, y, compute random secret share $[x \wedge y]$ s.t. neither party learns $x \wedge y$



y

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

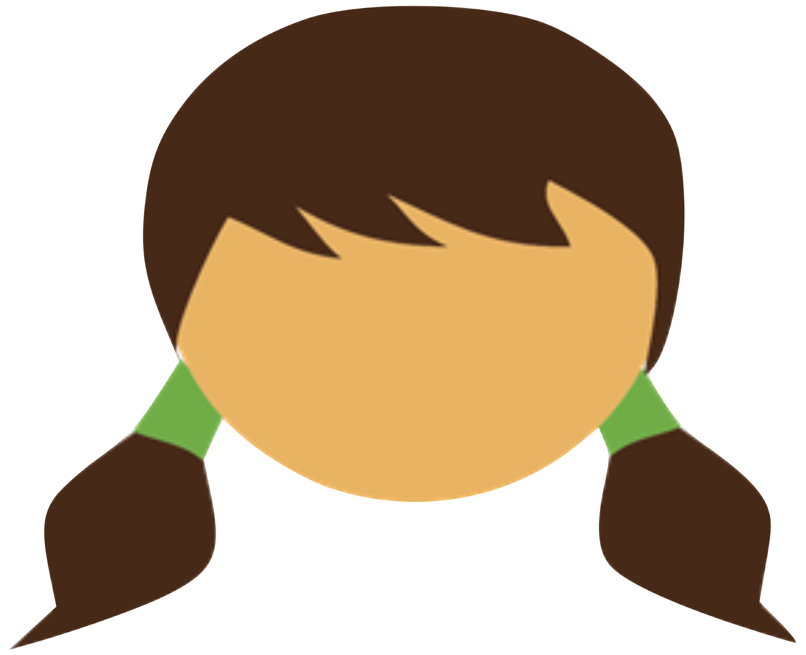
$r, r \oplus x$ $\qquad$ y



x

Important Subgoal

Goal: given gate input bits x, y, compute random secret share $[x \wedge y]$ s.t. neither party learns $x \wedge y$

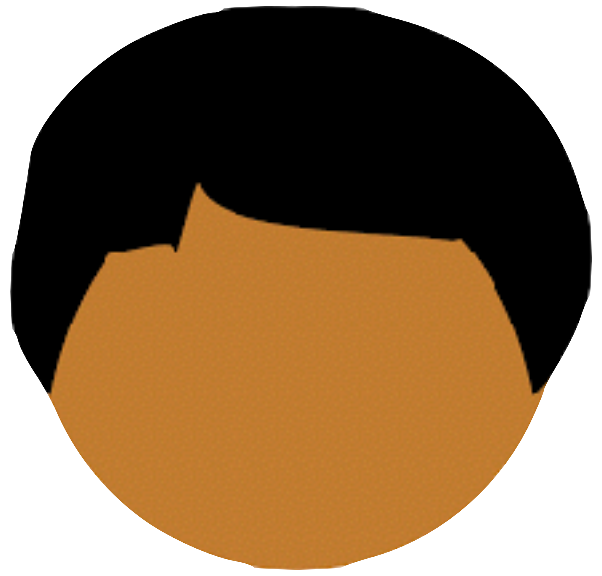


y

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

$r, r \oplus x$ → **OT** ← y

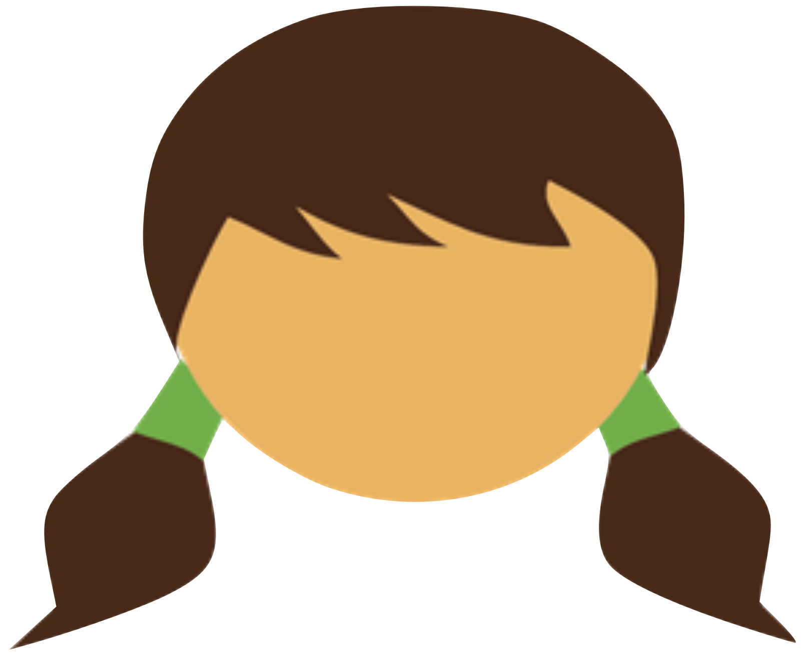
OT → $r \oplus (x \wedge y)$



x

Important Subgoal

Goal: given gate input bits x, y, compute random secret share $[x \wedge y]$ s.t. neither party learns $x \wedge y$



y

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

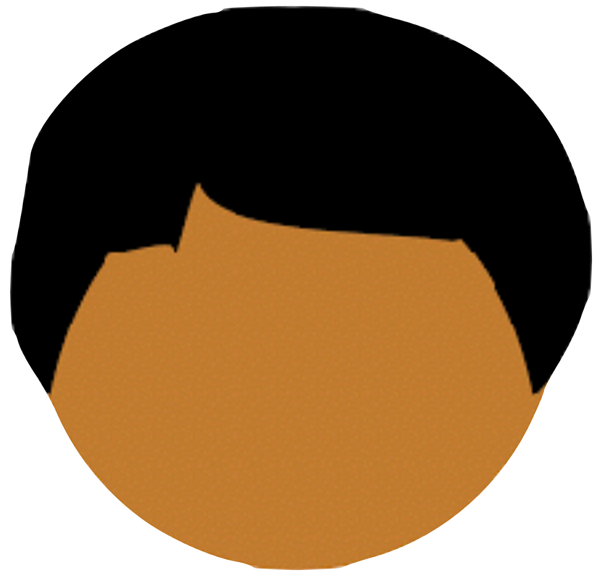
$r, r \oplus x$

y

OT

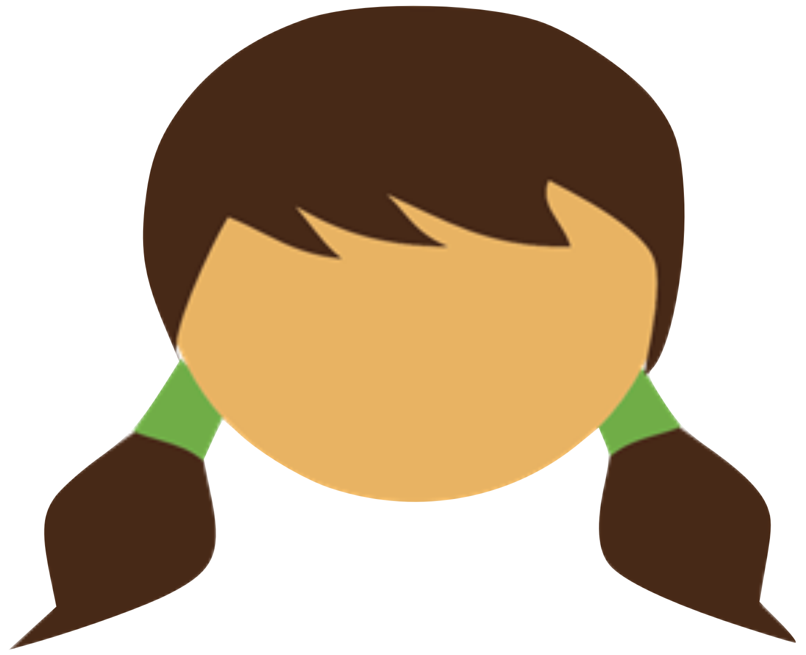
$\langle r, r \oplus (x \wedge y) \rangle = [x \wedge y]$

$r \oplus (x \wedge y)$



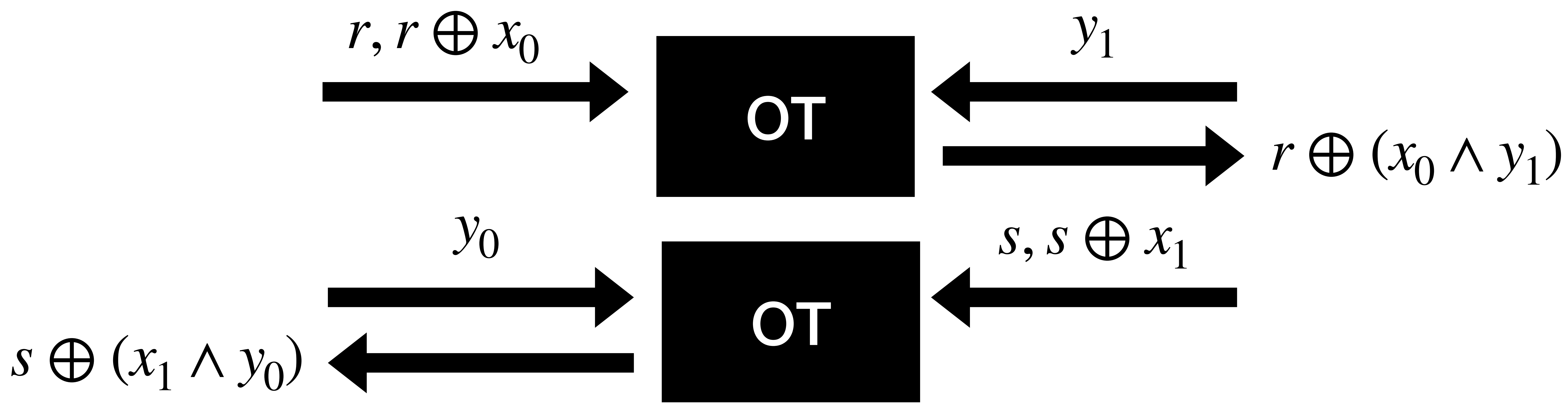
How do we AND two shares?

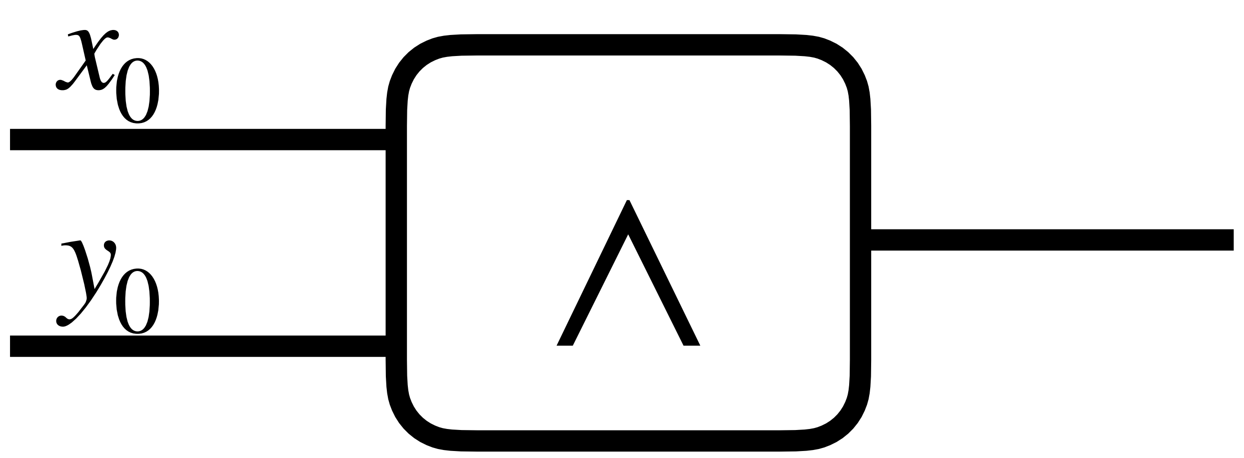
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \wedge y]$ on the gate output

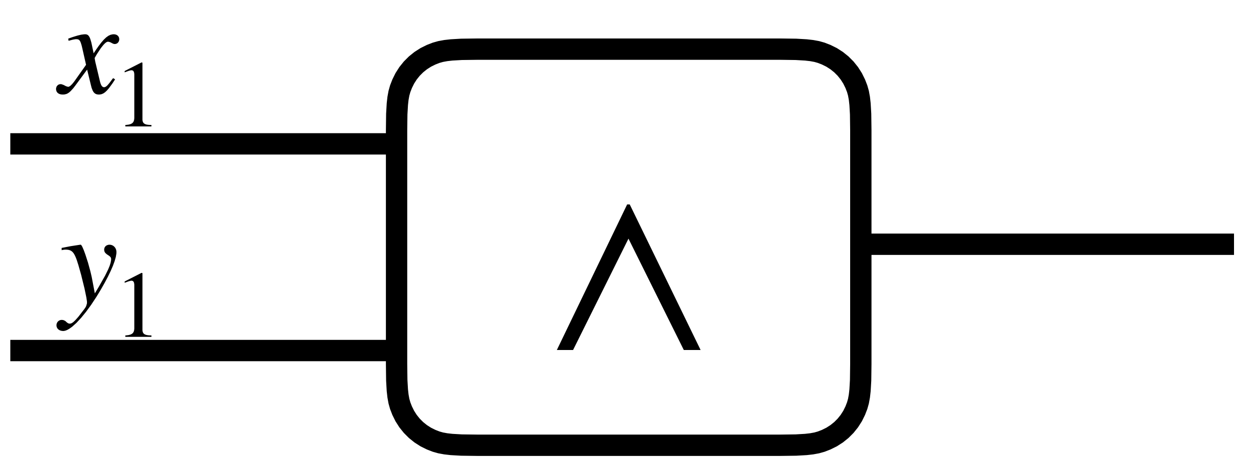


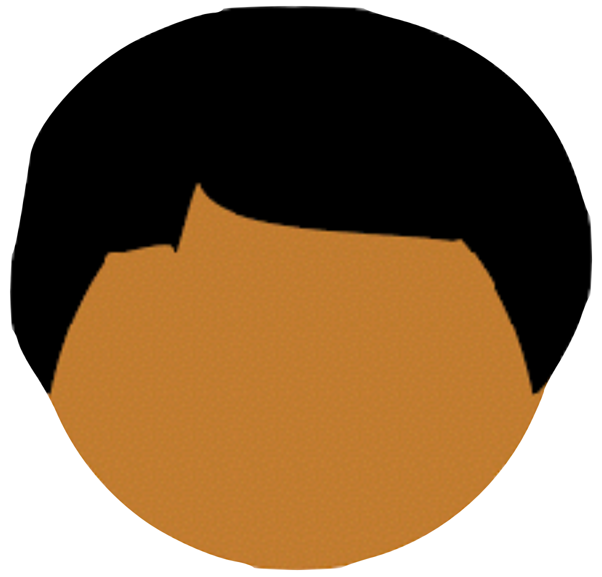
$r \xleftarrow{\$} \{0,1\}$

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

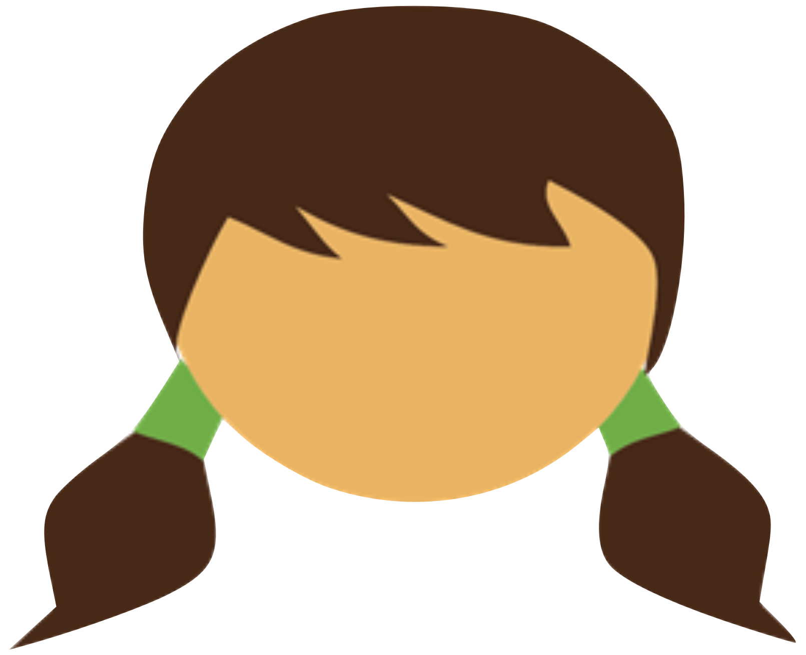






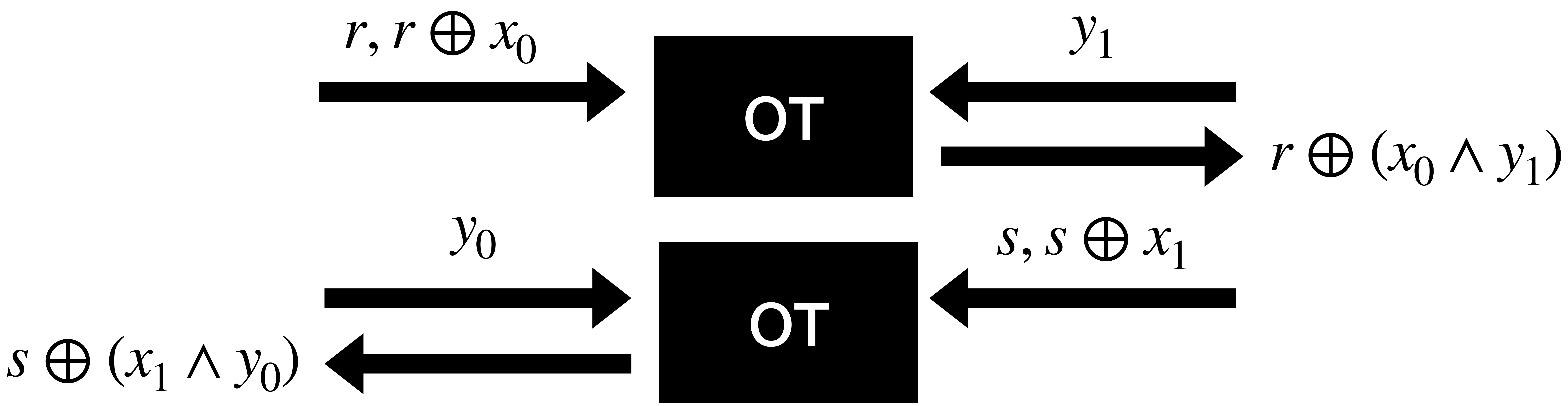


How do we AND two shares?



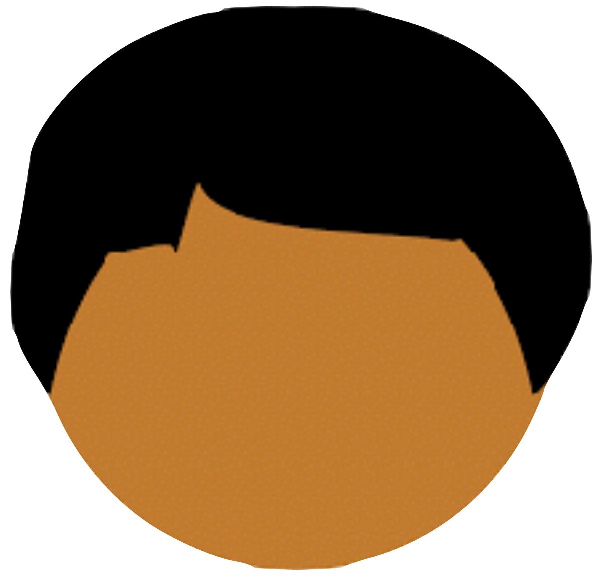
Goal: given gate input wires holding $[x]$, $[y]$, put $[x \wedge y]$ on the gate output

$r \xleftarrow{\$} \{0,1\}$ $\qquad$ $s \xleftarrow{\$} \{0,1\}$



$$\langle r \oplus (s \oplus x_1 \wedge y_0) \oplus (x_0 \wedge y_0), s \oplus (r \oplus x_0 \wedge y_1) \oplus (x_1 \wedge y_1) \rangle$$

$$= [x \wedge y]$$

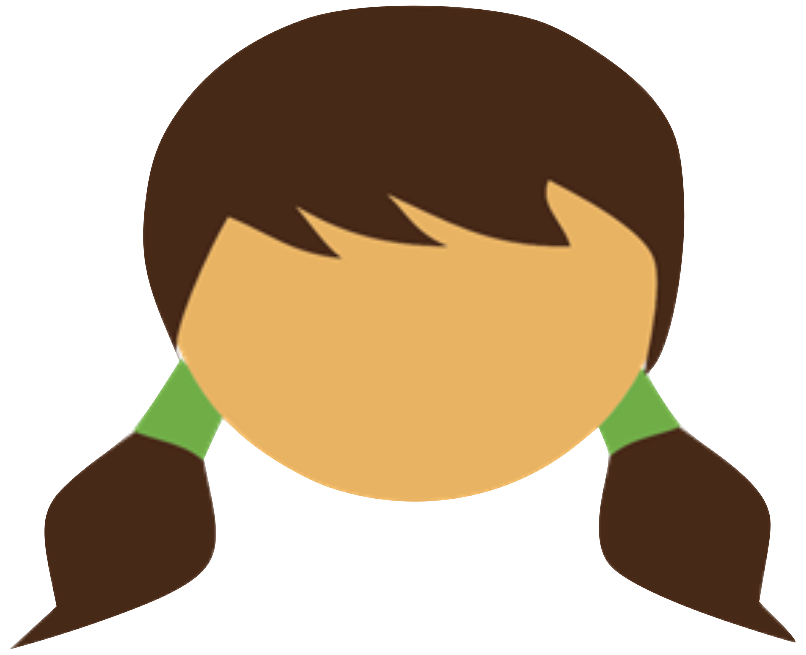


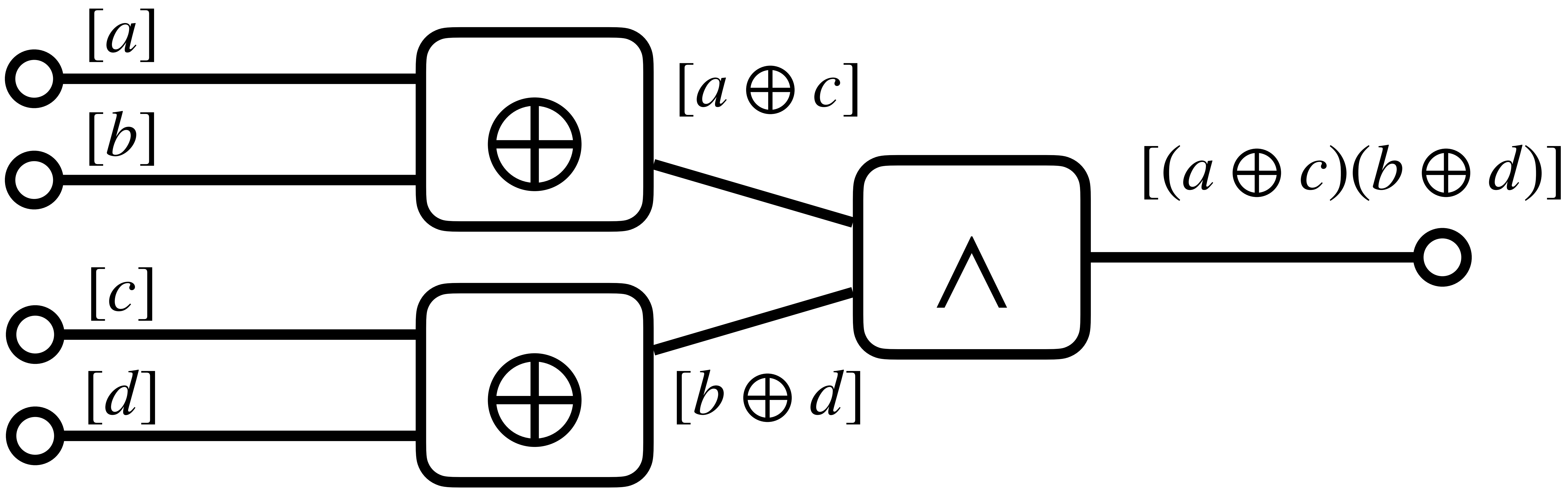
Where do input shares come from?

How do we XOR two shares?

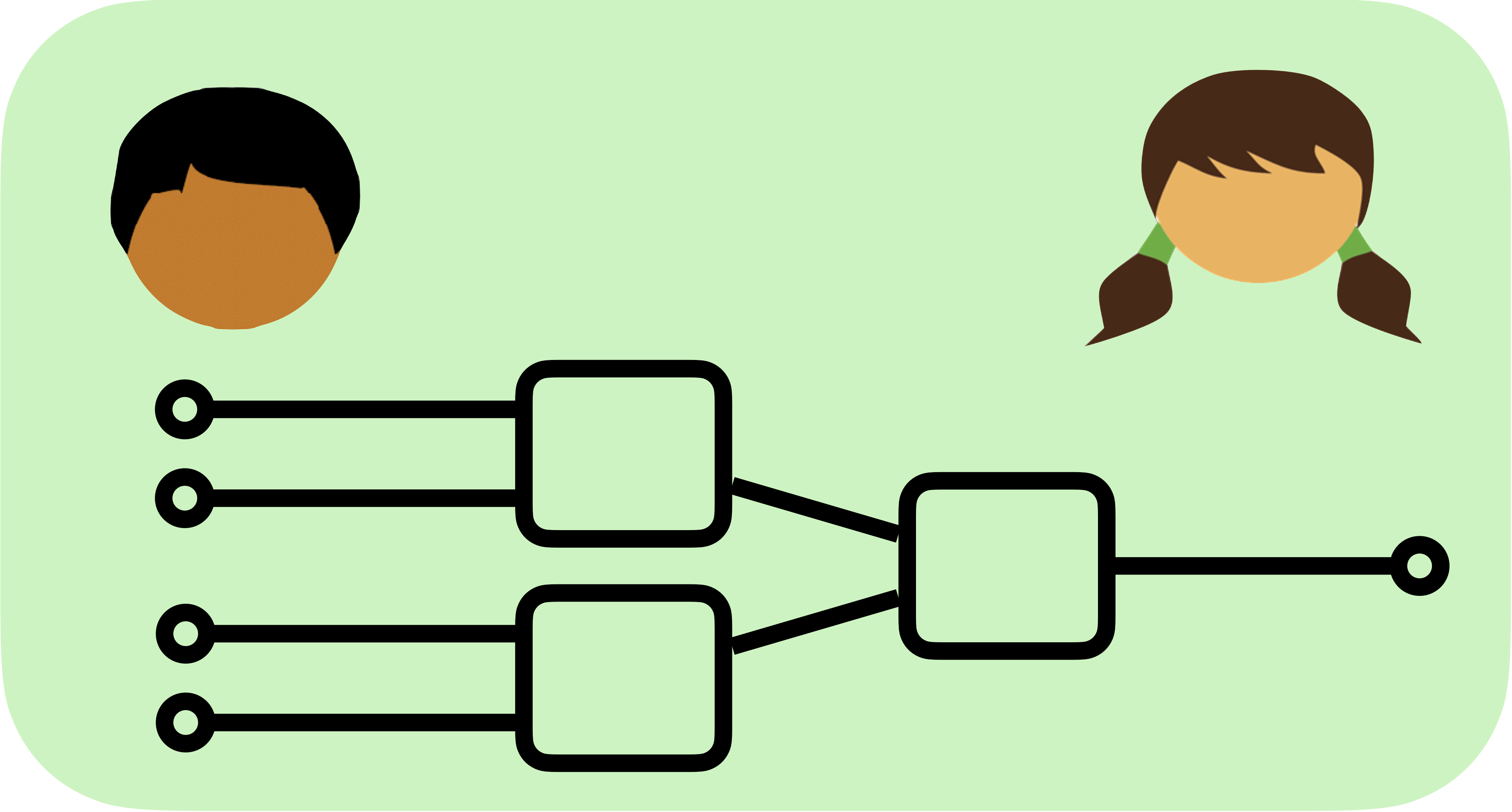
How do we AND two shares?

How do we “decrypt” output shares?





GMW Protocol



Propagate secret shares from input wires to output wires

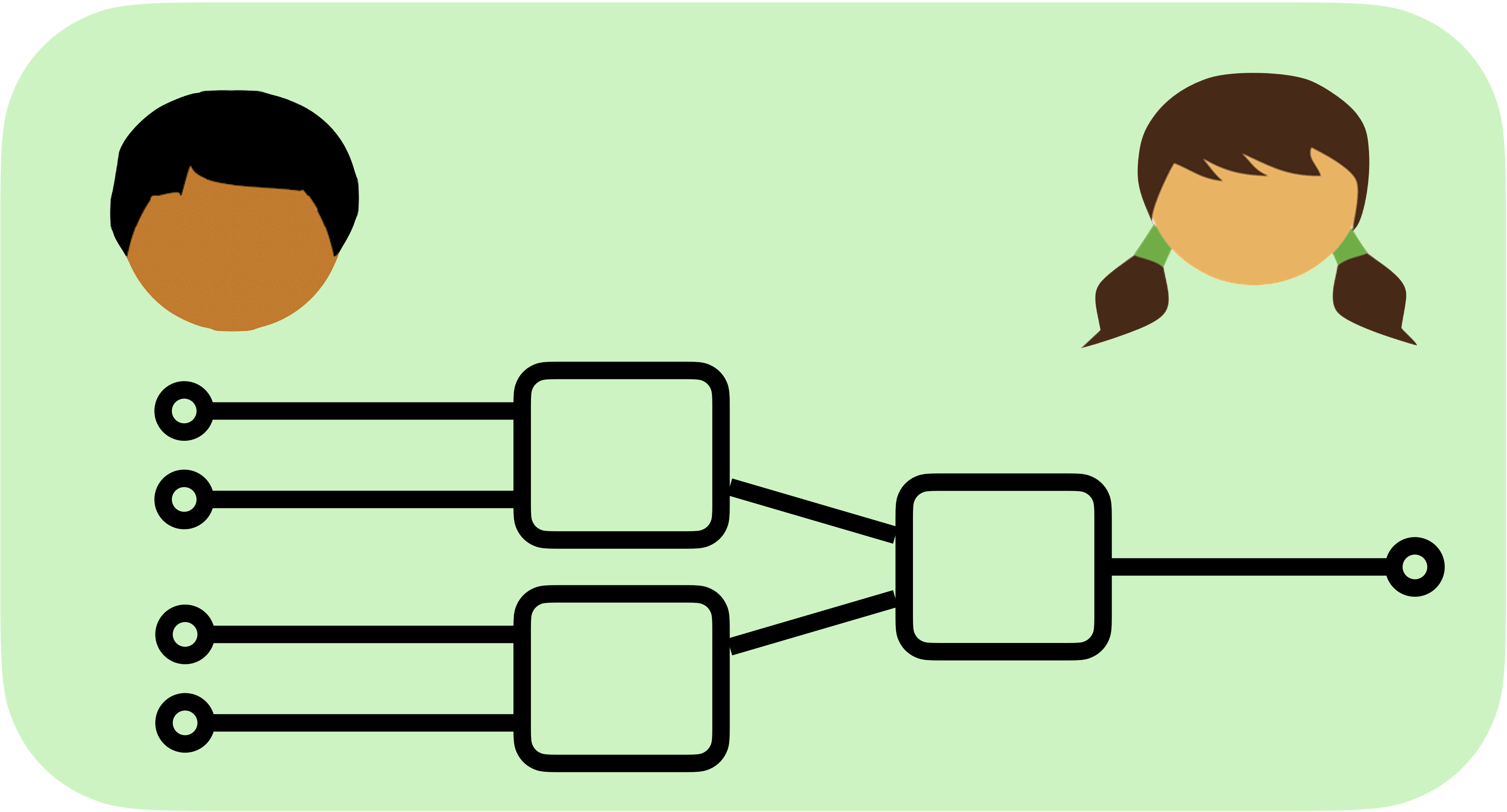
Use OT to implement AND gates

Cost:

$O(|C|)$ OTs

Number of protocol rounds scales with the **depth** of C

Now we know how to run any program



What is the MPC field about?

More Parties

Stronger Security Notions

Improved Efficiency