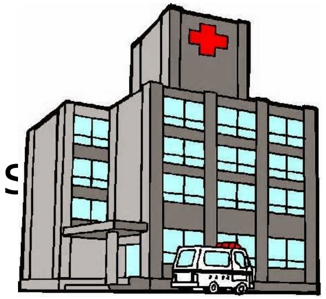
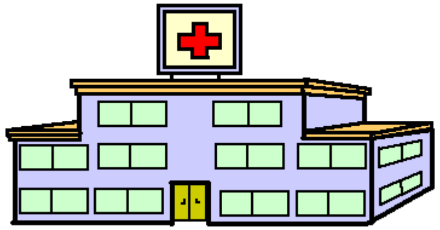
Secure Multi-party Computation

NISHANTH CHANDRAN
DIVYA GUPTA

Machine Learning on Health Records

Patient Records DB_1





Patient Records DB_2

Can hospitals compute joint
statistics on their databases
without revealing patient information to one another?

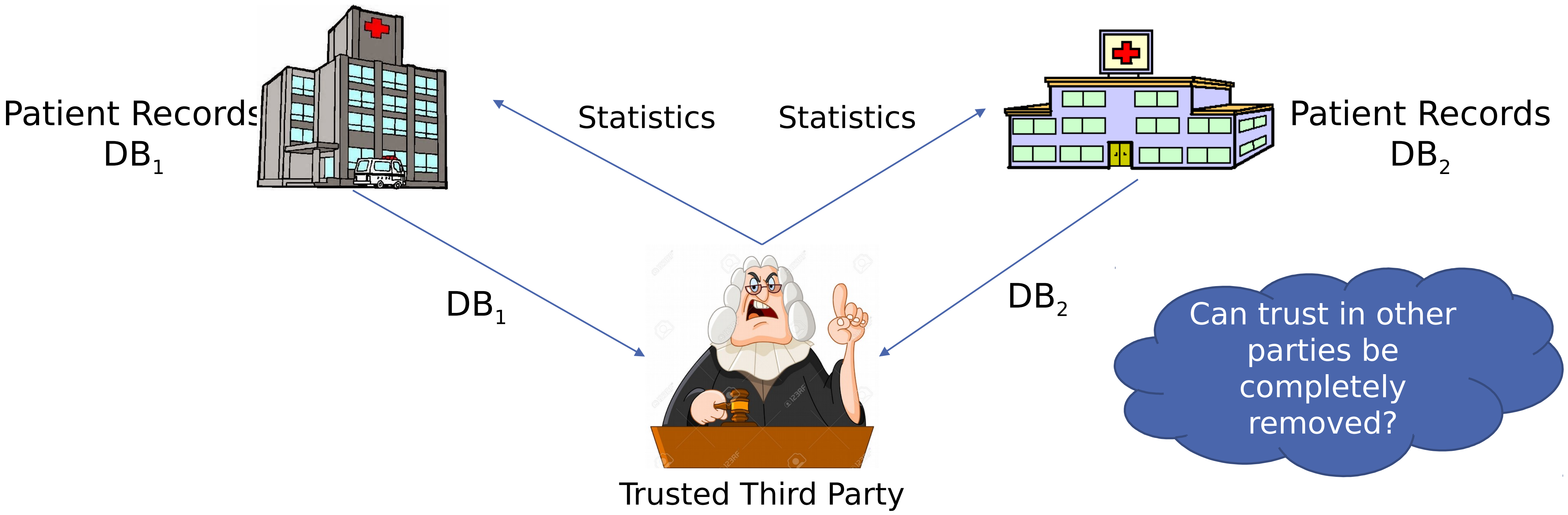
Private Set Intersection

$CustList_1$

$CustList_2$

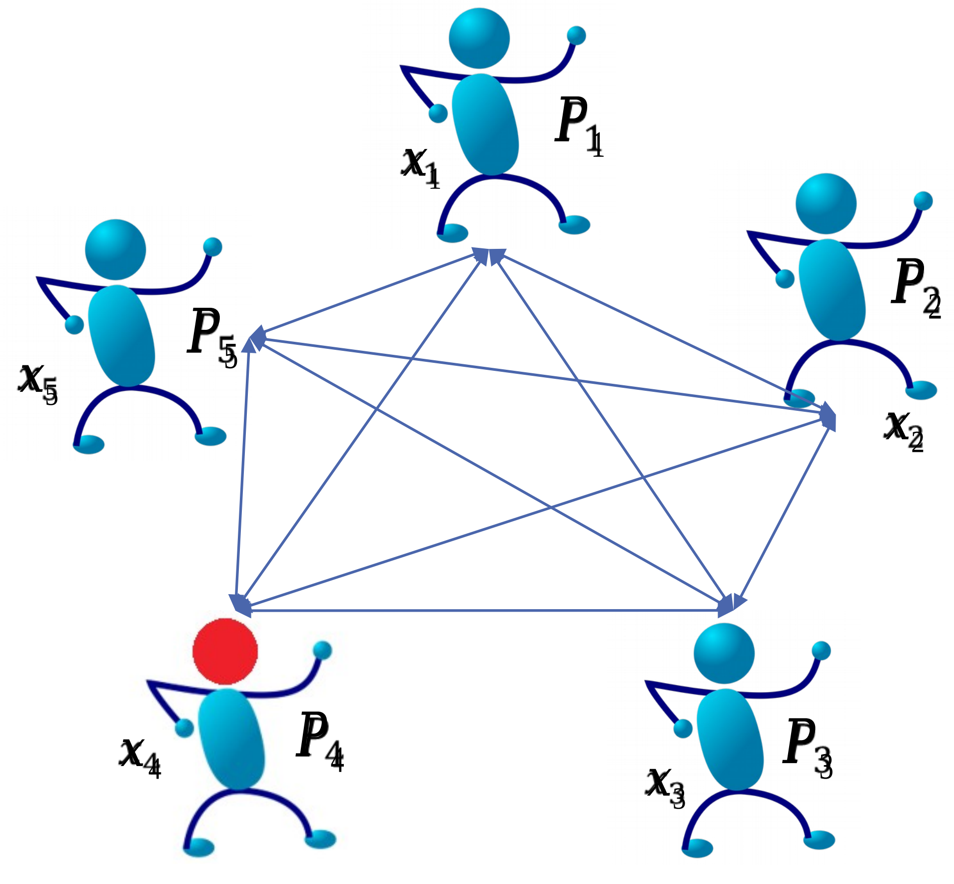
- Realty companies find list of customers who have double listed
- Can they do so without revealing individual customer names to each other?

A way to solve this problem



Secure Two/Multi-party Computation (MPC)

[Yao86, GMW87, BGW88, CCD88]



- n parties, t corruptions
- P_i has input x_i
- Goal is to compute $f(x_1, x_2, \ldots\ldots, x_n)$
- Correctness: Execute protocol to compute $f(x_1, x_2, \ldots, x_n)$ correctly
- Security: Parties should not learn anything* about other parties' inputs

Talk Outline

- What is security in 2PC/MPC?
- Boolean Computation: Yao’s 2-party Garbling protocol
- Arithmetic Computation: Secret sharing and Beaver Triplets
- EzPC: Making MPC usable

Two-party Computation Security

Alice should not learn anything* about Bob's input

Two-party Computation Security

Net worth:
X $

Net worth:
Y $

Two-party Computation Security

Net worth:
X $

Net worth:
Y $

$f(x,y)=x+y$

Alice should not learn anything* about Bob's input; *What does Alice learn?*

$f(x,y)=x+y$

Two-party Computation Security

Net worth:
X $

Secure Computation cannot prevent Alice from learning what she could have learned about Bob from the output (and her input)

Net worth:
Y $

Defining Security:
Alice *learns nothing more* than what can be learned from x and f(x,y)

Two-party Computation Security

Net worth:
X $

Who is richer?
(i.e., is X>Y ?)

Net worth:
Y $

Alice and Bob learn if X>Y but nothing more

Two-party Computation Security

Alice and Bob execute a protocol to compute f(x,y)

Net worth:
X $

Net worth:
Y $

Will Bob learn nothing about *x* (other than *f(x,y)*) even when he does not execute the protocol honestly?

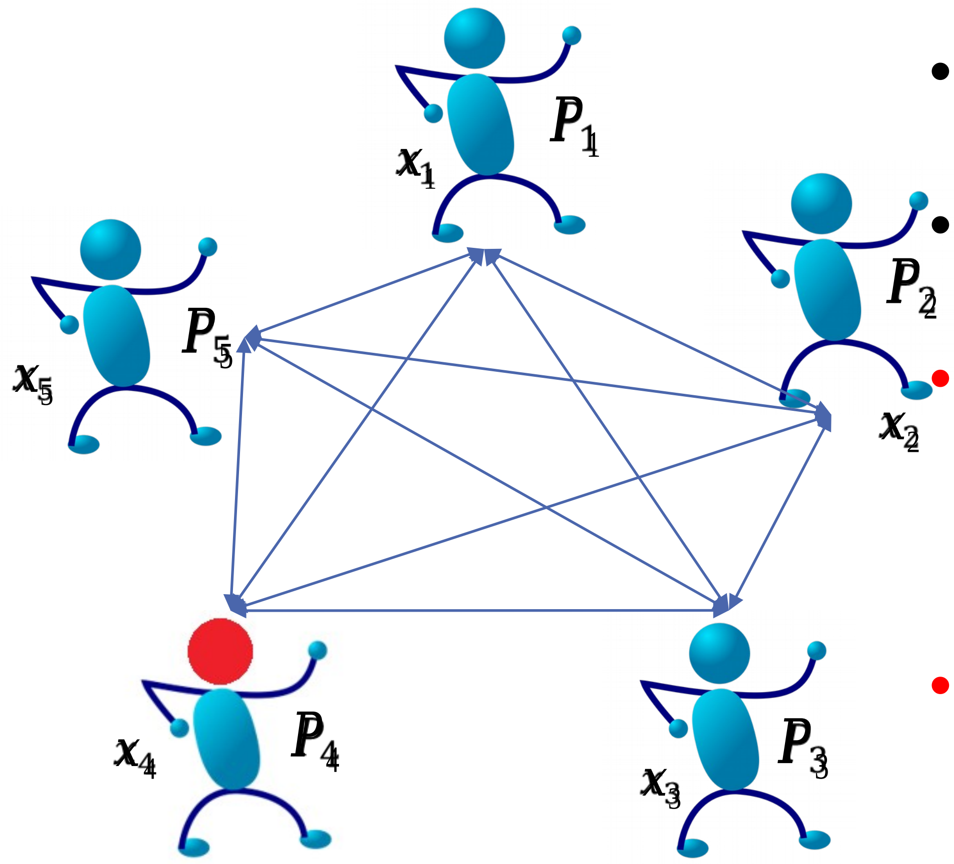
Two Kinds of Security – Semihonest vs Malicious

Net worth:
X $

Net worth:
Y $

Semihonest	Malicious
• Security guaranteed when malicious party follows the protocol honestly	• Security guaranteed even when malicious party does not follow the protocol honestly

Secure Multi-party Computation (MPC)



- Similar security notions
- Includes a corruption threshold $t < n$
- Semihonest: t parties colluding do not learn any more information when they all follow the protocol honestly
- Malicious: t parties colluding do not learn any more information even when they do not follow the protocol

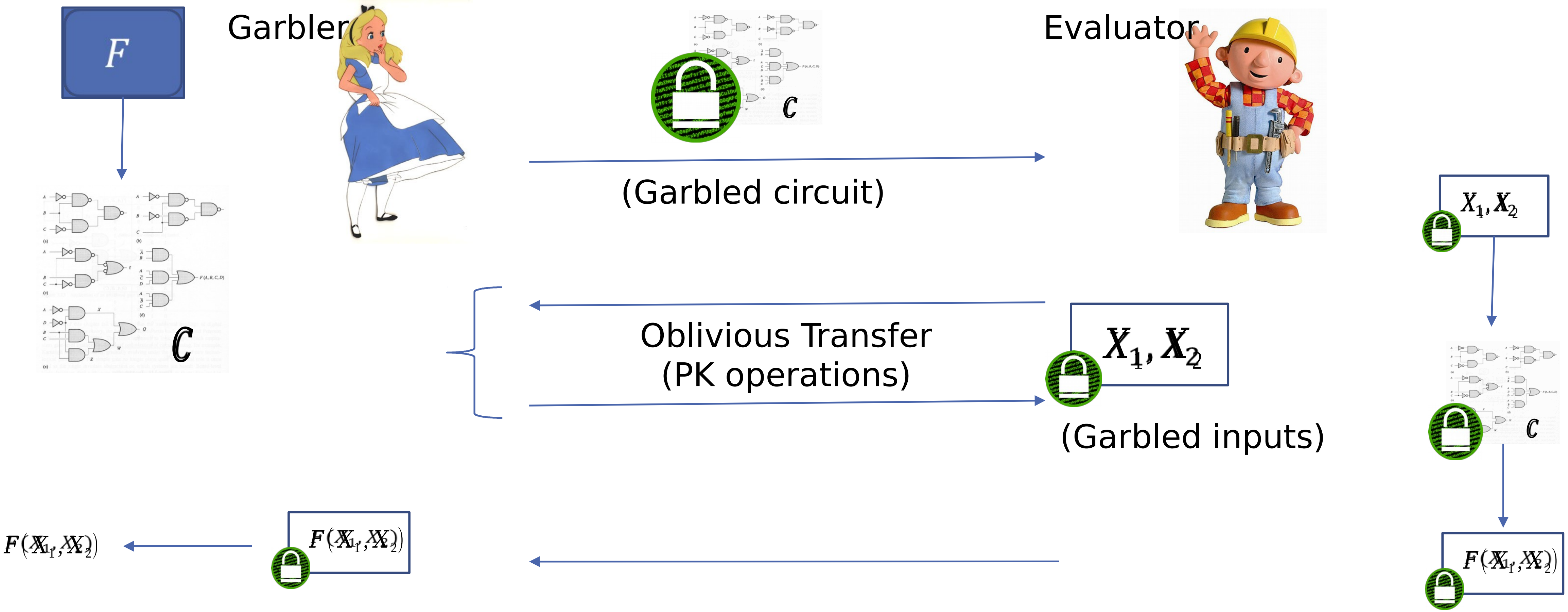
Talk Outline

- What is security in 2PC/MPC?
- Boolean Computation: Yao's 2-party Garbling protocol
- Arithmetic Computation: Secret sharing and Beaver Triplets
- EzPC: Making MPC usable

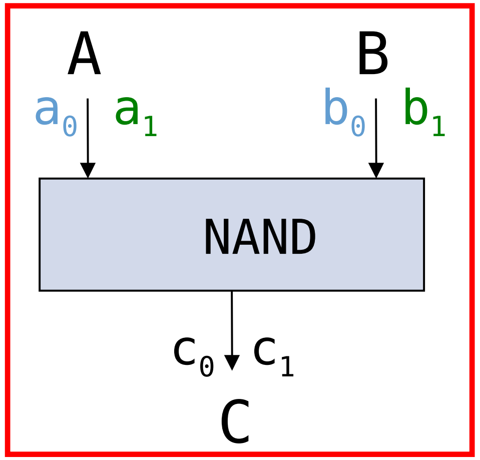
Boolean Computation

- All compute expressed as Boolean circuits (AND, XOR gates)
- Comparison, Bit-shifts etc. are most efficient when expressed as Boolean circuits
- Multiplication costs O(l^2)

Technique for 2 PC – Garbled Circuits [Yao86]



How to Garble a gate? (E.g. NAND)



- Alice picks 2 random keys per wire (6 per gate).
- One key corresponds to 0 and the other to 1.
  - If A = 0, then key = a_0.

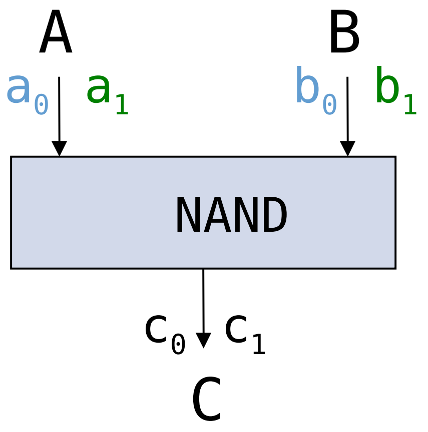
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NAND Gate Truth Table

A	B	C
a_0	b_0	c_1
a_0	b_1	c_1
a_1	b_0	c_1
a_1	b_1	c_0

Truth Table with Keys

How to Garble a gate? (E.g. NAND)



- Alice picks 2 random keys per wire (6 per gate).
- One key corresponds to 0 and the other to 1.
  - If A = 0, then key = a_0.

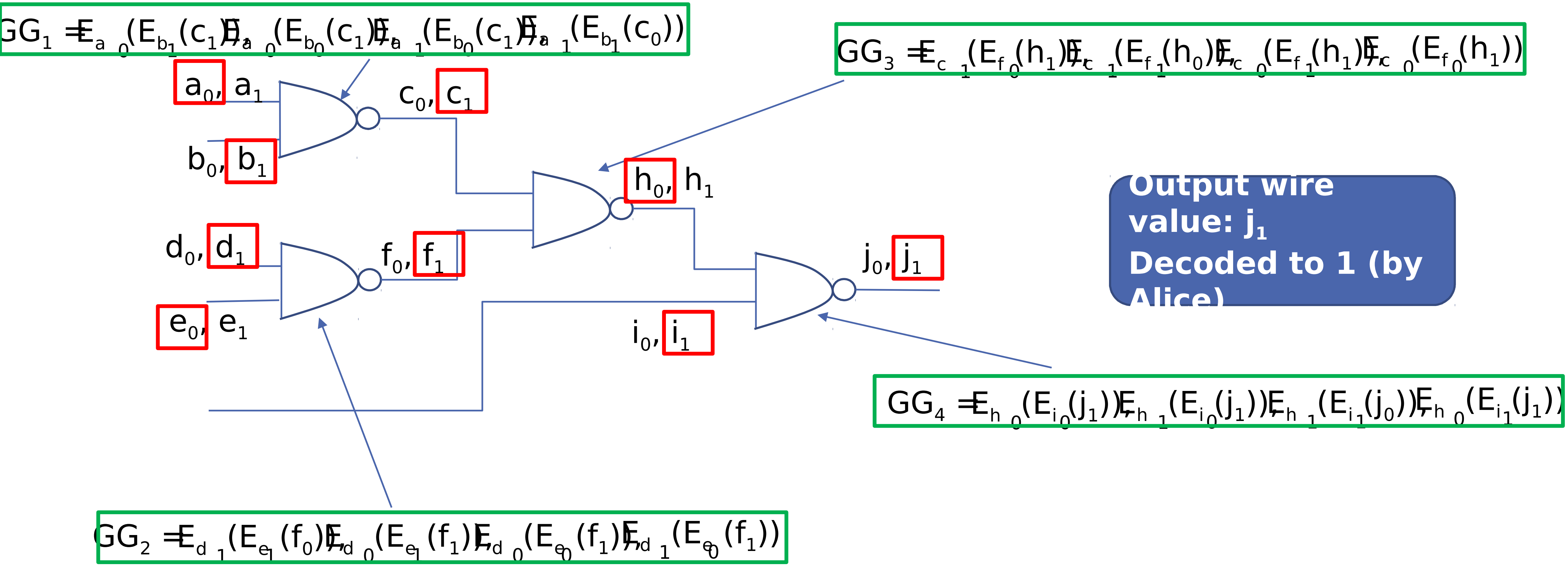
How does Bob evaluate it?

A	B	C	Garbled NAND Gate
a_0	b_0	c_1	$E_{a_0}(E_{b_0}(c_1))$
a_0	b_1	c_1	$E_{a_0}(E_{b_1}(c_1))$
a_1	b_0	c_1	$E_{a_1}(E_{b_0}(c_1))$
a_1	b_1	c_0	$E_{a_1}(E_{b_1}(c_0))$

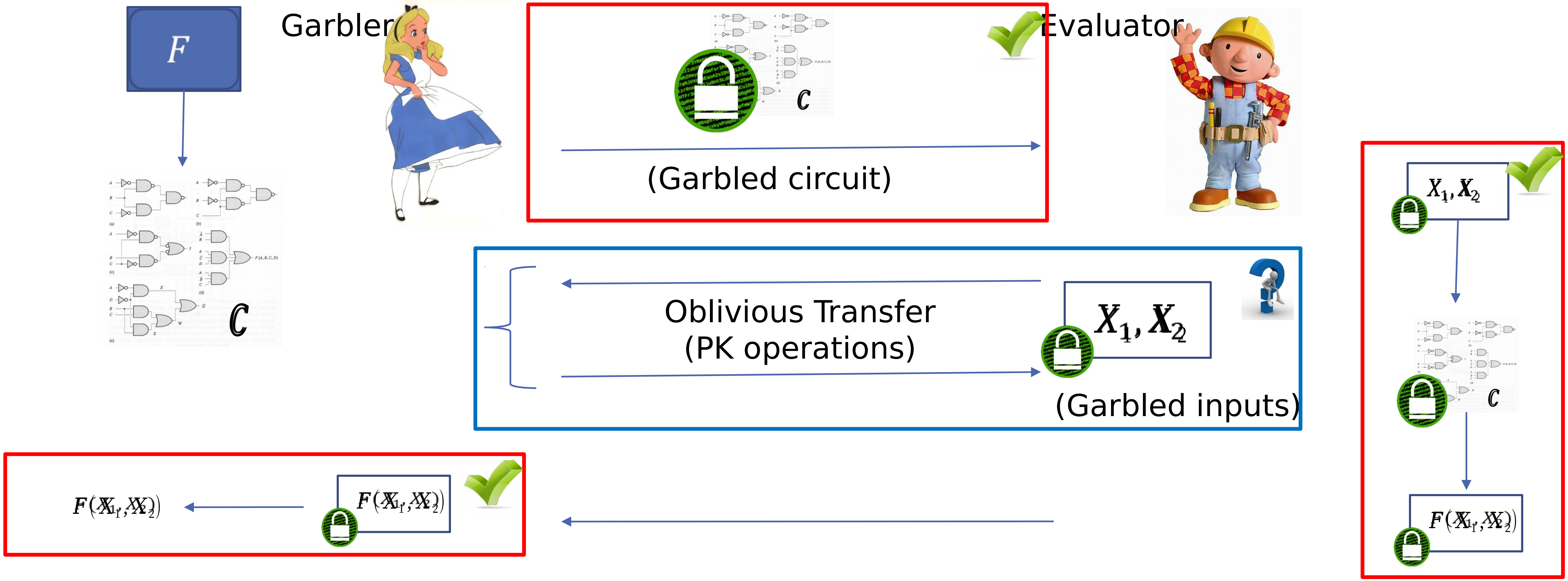
Truth Table with Keys

This ciphertext alone will decrypt correctly.

How does a Garbled Circuit look?



Technique for 2 PC – Garbled Circuits [Yao86]



Oblivious Transfer [Rabin81, EGL85]

m_0, m_1

b

Learns m_b

- Security 1: Alice does not learn b
- Security 2: Bob does not learn $m_{1\text{-}b}$

A protocol for Oblivious Transfer (OT) from (special) public-key encryption

m_0, m_1

(pk_0 , pk_1)

b

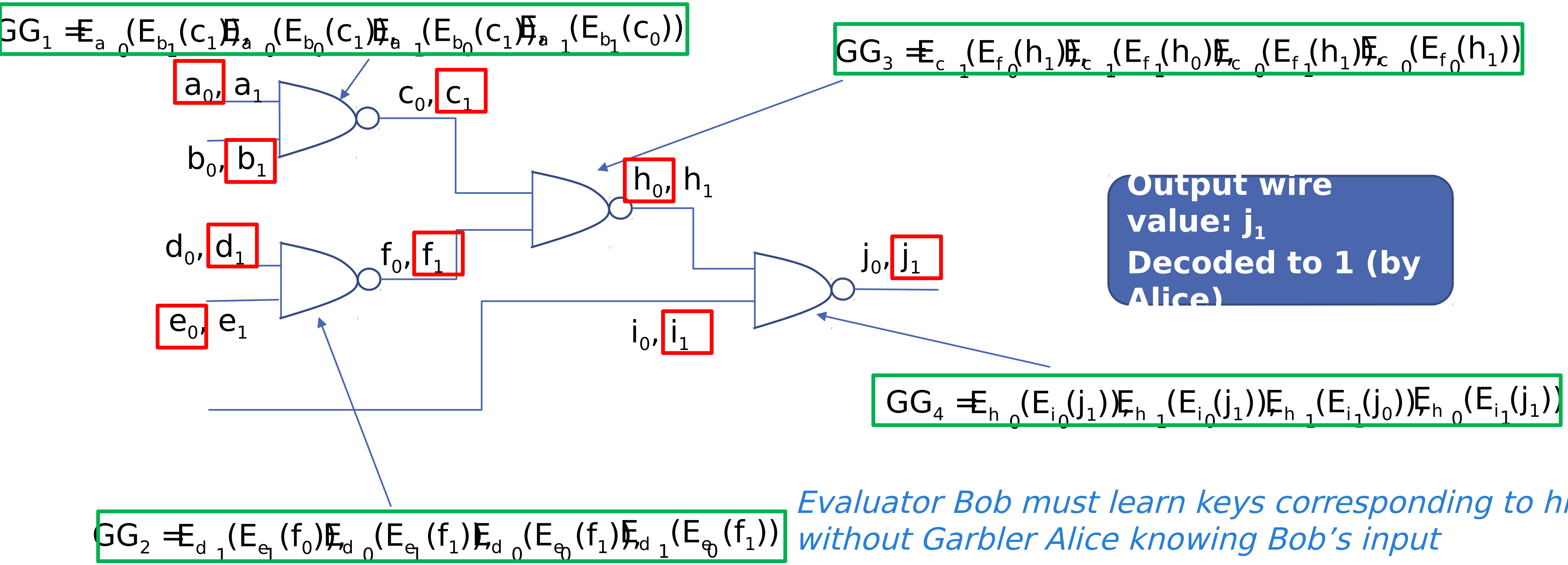
c_0 = $\mathrm{Enc}_{pk0}(m_0)$, c_1 = $\mathrm{Enc}_{pk1}(m_1)$

- Pick (pk_b, sk_b) and pk_{1-b}
- Decrypt c_b to learn m_b

Security 1: Alice does not learn b because pk_0 and pk_1 are indistinguishable

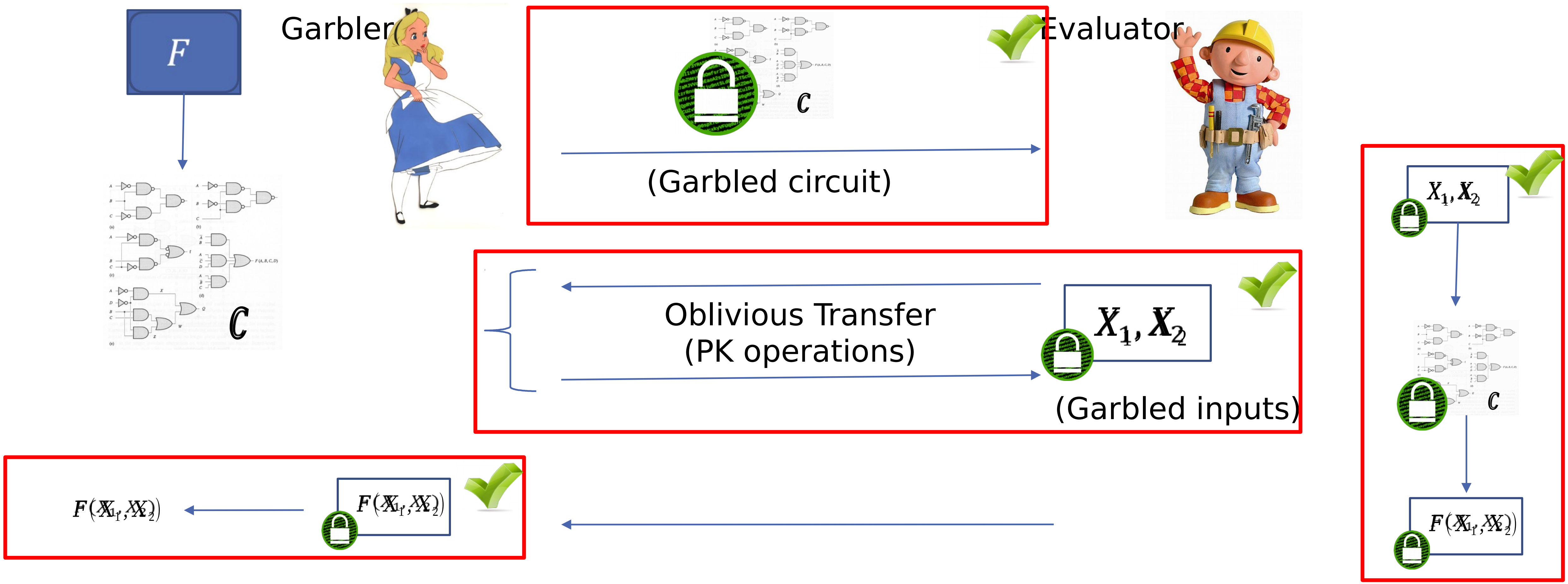
Security 2: Bob does not learn m_{1-b} because he does not know sk_{1-b}

Where do OTs fit in Garbled Circuits?



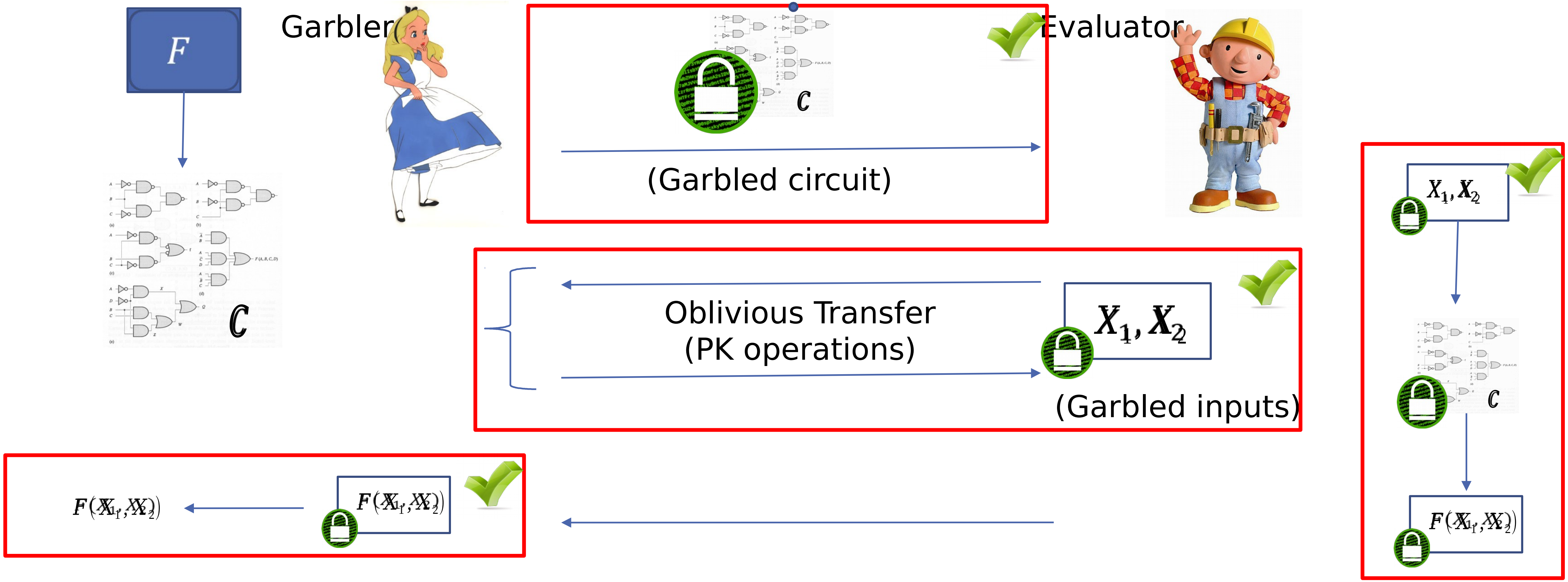
Evaluator Bob must learn keys corresponding to his input without Garbler Alice knowing Bob's input

Putting it all together



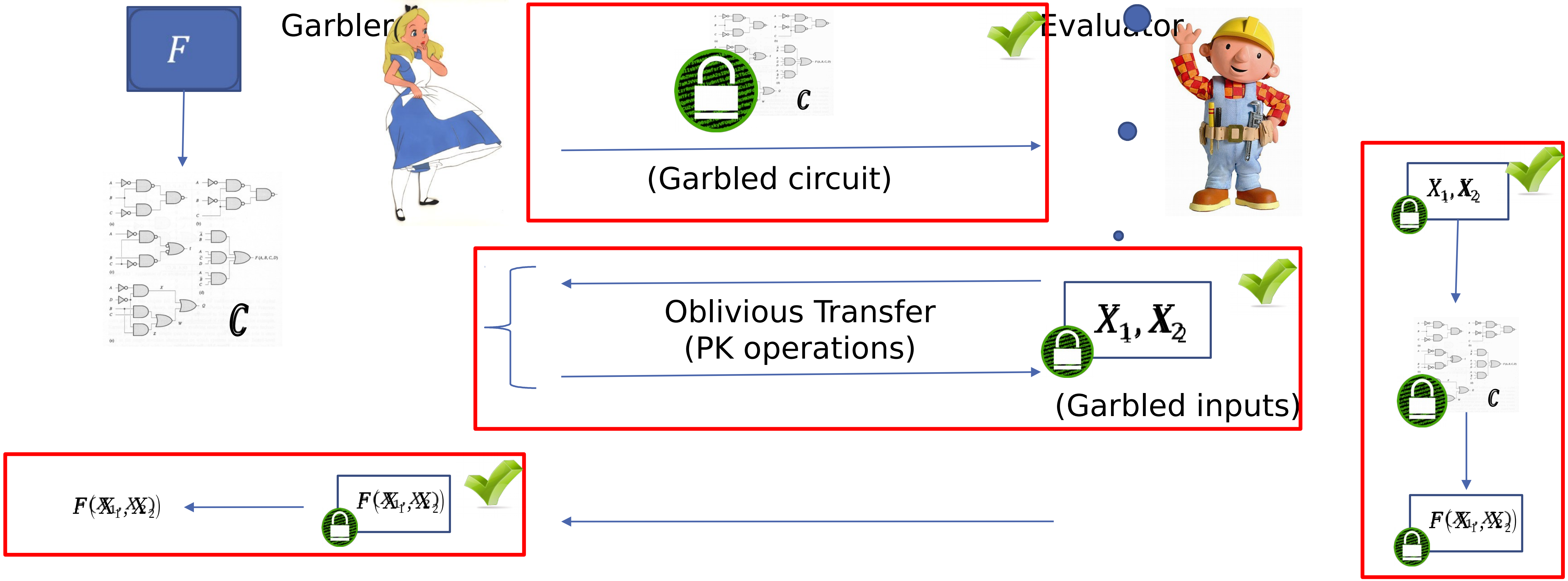
Why is the protocol se… easy)

Only depends on C - No information about Alice's input



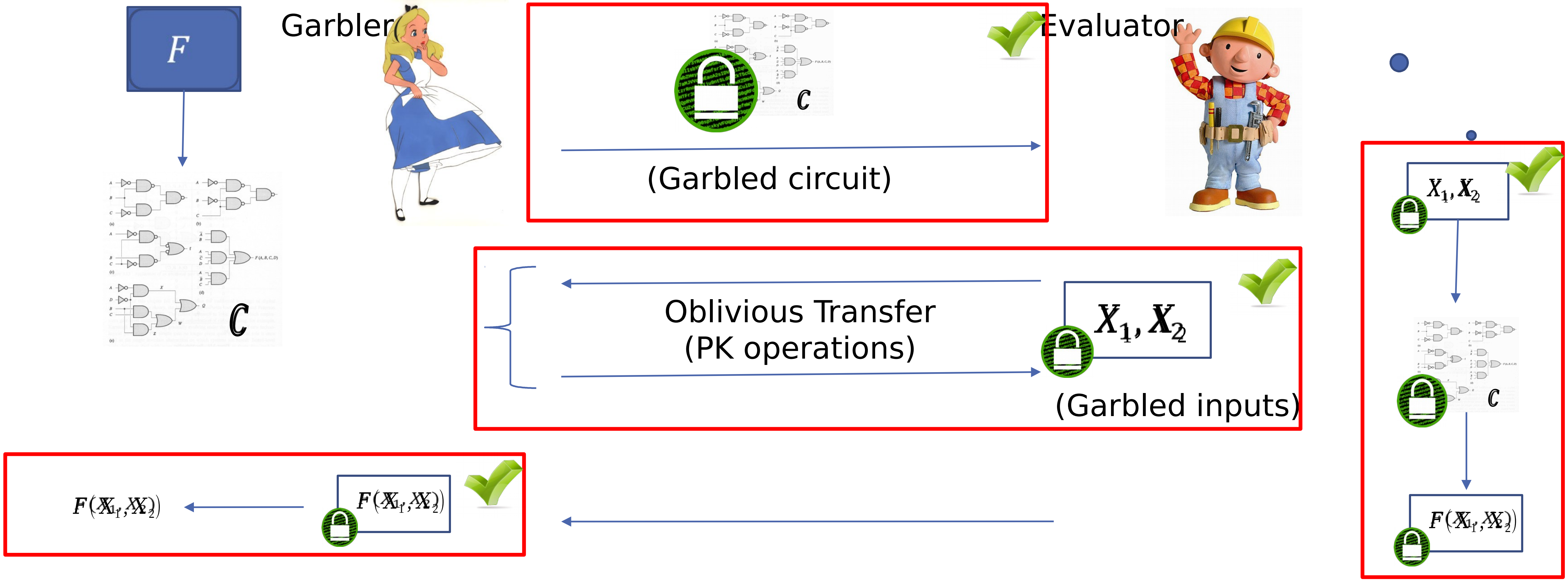
Why is the protocol … easy)

OT security says Alice does not learn Bob's input and Bob learns only one key



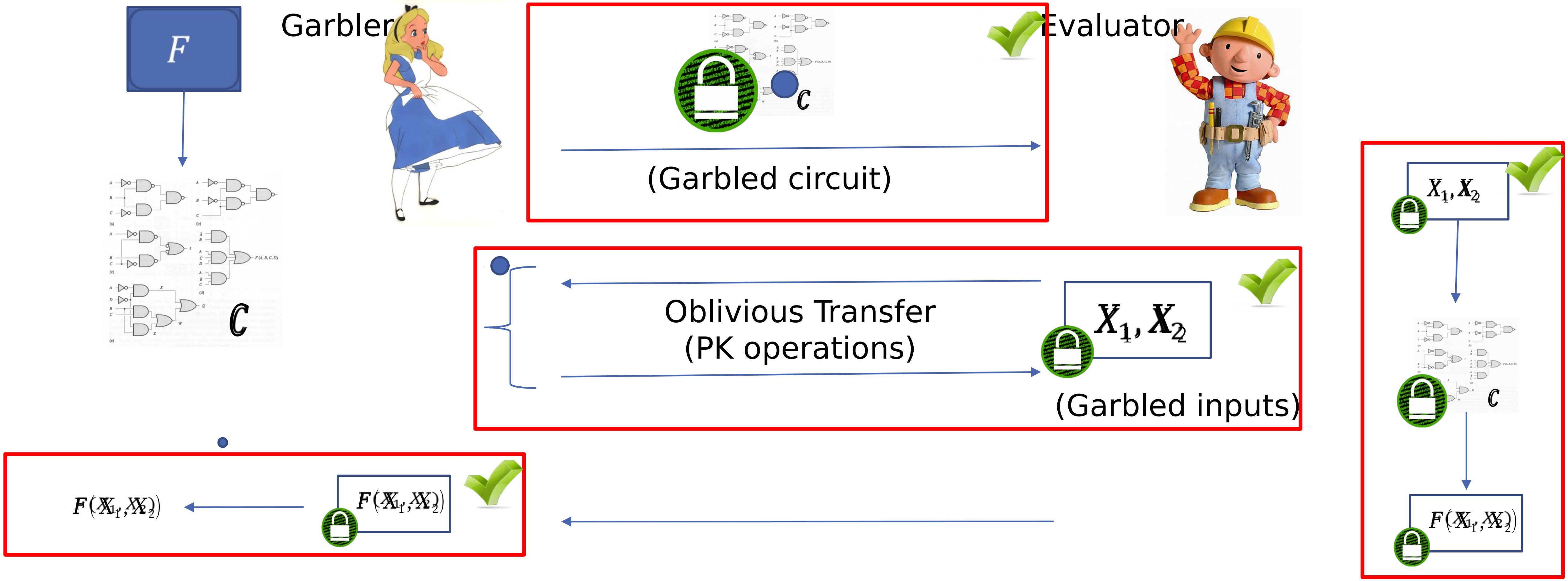
Why is the protocol
easy)

(Tricky) proof can show that Bob only learns one final key and no other information

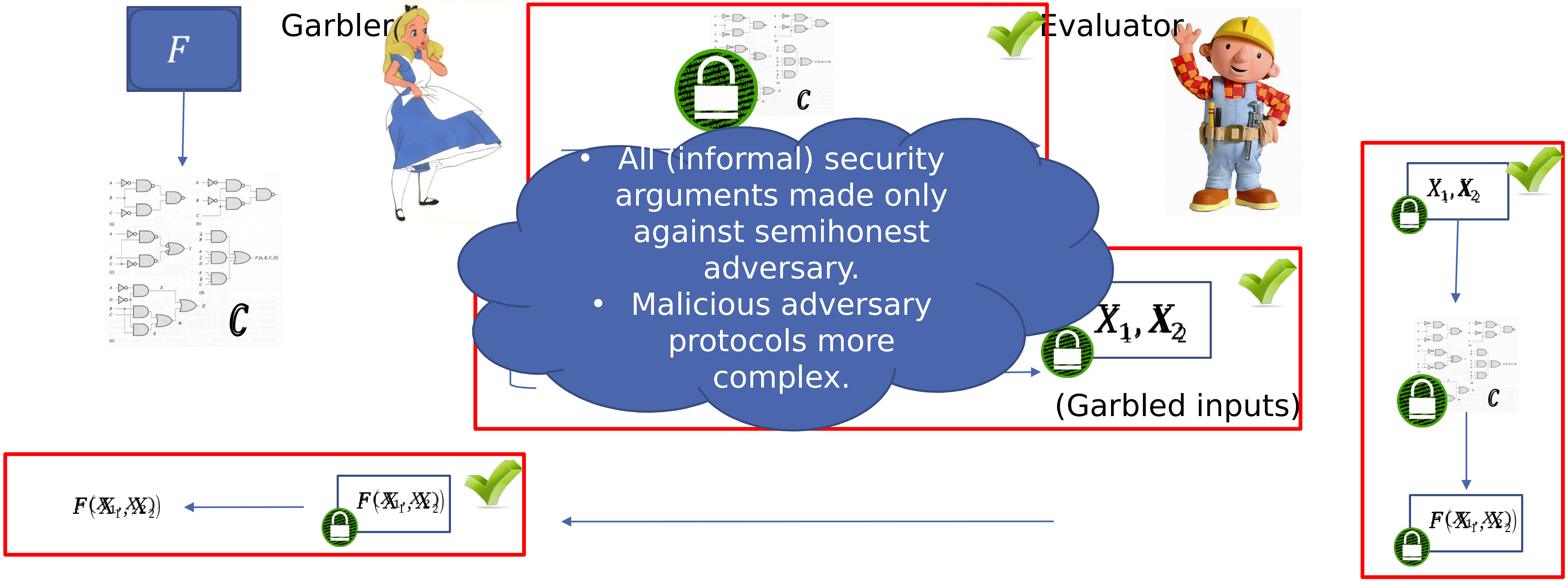


Why is the protocol … easy)

Alice only sees one final key corresponding to $f(x_1, x_2)$



Why is the protocol secure? (Not easy)



Talk Outline

- What is security in 2PC/MPC?
- Boolean Computation: Yao's 2-party Garbling protoc
- Arithmetic Computation: Secret sharing and Beaver Triplets
- EzPC: Making MPC usable

Secure Multi-party Computation & Applications to Private Machine Learning

DIVYA GUPTA

Microsoft Research

[1] THE VIEWS EXPRESSED HERE ARE SOLELY THOSE OF THE AUTHOR IN HIS PRIVATE CAPACITY AND DO NOT IN ANY WAY REPRESENT THE VIEWS OF MICROSOFT, OR ANY OTHER ENTITY OF MICROSOFT.

Talk Outline

- Secure Computation for Arithmetic Circuits

- EzPC: Programmable, Efficient, and Scalable Secure Computation

(applied to private machine learning)

Secure Computation of Arithmetic Circuits

Input $x \in Z_{2^{32}}$ | Input $y \in Z_{2^{32}}$

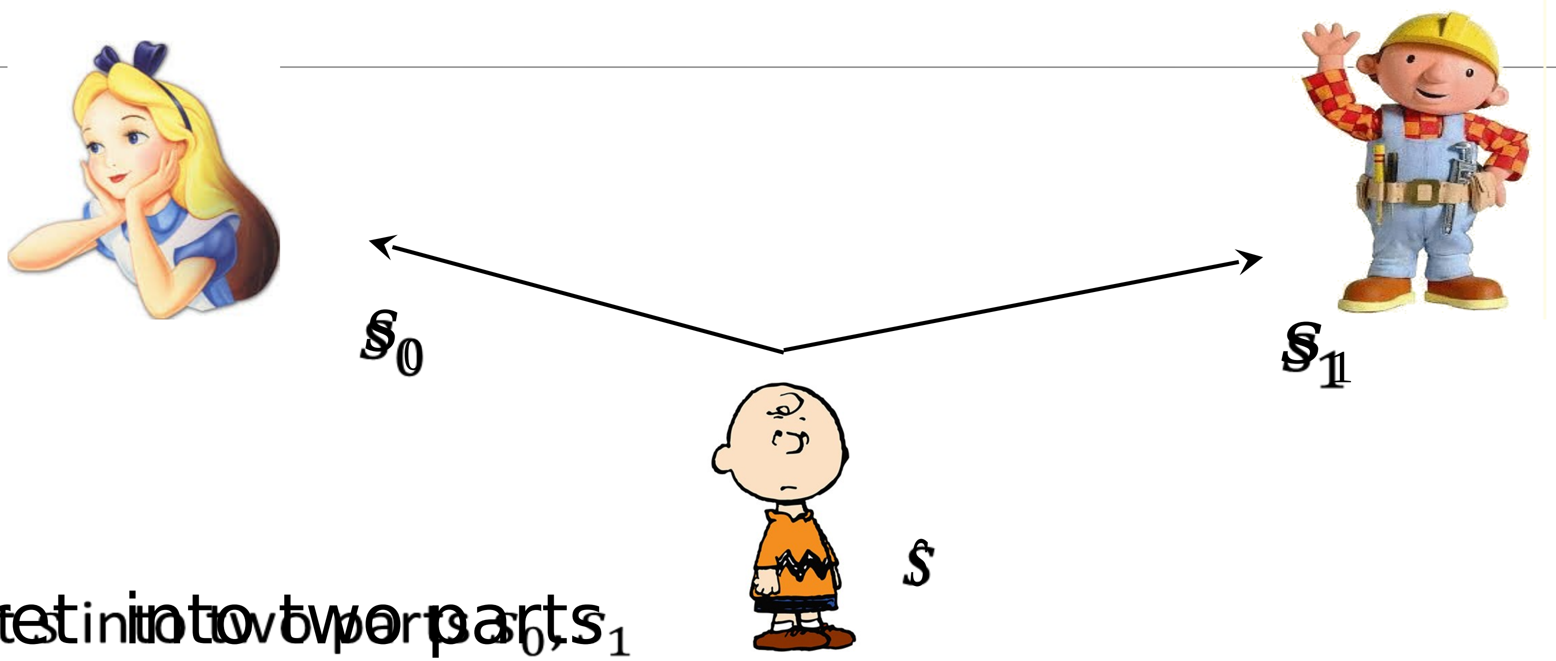
F = Arithmetic circuit

- Arithmetic circuits have addition and multiplication gates

Protocol Summary:

- Alice and Bob start with 2-out-2 secret shares of input
- For a gate, given shares of input wires, run a protocol to compute shares of output wire

2-out-of-2 Secret sharing scheme



- Split secret s into two parts s_0, s_1
- Single share reveals nothing about s
- Combine shares to get s

- **Example: Uniform shares s.t. $s_0 + s_1 = s$**

Input sharing phase

- Each party shares its input with other party

Input $x \in Z_{2^{32}}$

x_1 →

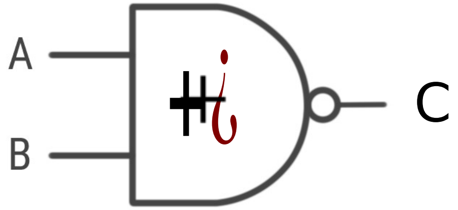
← y_0

Input $y \in Z_{2^{32}}$

Pick x_0, x_1 s.t. $x_0 + x_1 = x$

Pick y_0, y_1 s.t. $y_0 + y_1 = y$

Addition gate



- Each locally adds the shares of input

a_0, b_0

No Interaction

a_1, b_1

Compute $c_0 = a_0 + b_0$ Compute

Compute $c_1 = a_1 + b_1$ Compute

- Correctness: $c_0 + c_1 = (a_0+b_0) + (a_1 + b_1) = a + b = c$
- Correctness:
- Security: trivial
- Security: trivial

Multiplication gate

x_0, y_0, z_0 * c

- Need setup such as "Beaver Triplet"
  - Parties hold shares of random x, y, z with $z = x * y$

- Need setup such as "Beaver Triplet"
- Parties hold shares of random with

x_0, y_0, z_0

a_0, b_0

$e_0 = a_0 - x_0, f_0 = b_0 - y_0$

Independent of input and circuit!

$e_1 = a_1 - x_1, f_1 = b_1 - y_1$

x_1, y_1, z_1

a_1, b_1

Reconstruct e and f

Compute

$c_0 = f * a_0 + e * b_0 + z_0$

Compute

$c_1 = f * a_1 + e * b_1 + z_1 - e * f$

Compute

Compute

Correctness?

Security?

Secure Computation Protocols

- **Boolean circuits:** Garbled circuits [Yao], GMW, BMR, …..

(Good for expressing comparisons, bitwise operations, maximum, etc)

- **Arithmetic circuits:** Using beaver triplets [Beaver], BGW, CCD, SPDZ, …..

(Good for e … ns and additions)

Very hard for non-crypto experts to select a good protocol for application!

Might need a mix of protocols!

Talk Outline

- Secure Computation for Arithmetic Circuits
- EzPC: Programmable, Efficient, and Scalable Secure Computation (applied to private machine learning)

Joint work with Nishanth Chandran, Aseem Rastogi and Rahul Sharma

Many Challenges in using 2PC

Function F

- Very hard for developers to write *secure* 2PC applications

- Which protocol is best suited for my application?
  - GMW, Yao, BGW, BMR, ……

- How to express the function efficiently?
  - Circuits: Boolean vs Arithmetic

- Most protocols require low circuit level programming
  - Tedious and error-prone

Our Goal: Democratizing 2PC

Make 2PC accessible to developers

Function $F(x, y)$

- Programmer-friendly platform
  - Developer *only* specifies functionality
- Generality: Express arbitrary functionalities
- Performance: Automatically choose right circuit rep.
- Scale to practical tasks
- Formal guarantees of Correctness and Security

Current state of affairs

Alice: w, b

Bob: x

Function: $w^t x > b$

Option 1

- Program in one of the several DSLs such as Fairplay, Wysteria, ObliVM, CBMC-GC, SMCL, Sharemind, etc
- Pro: High-level programmer friendly framework
- Pro: Developer is oblivious of underlying crypto magic
- Cons: **Poor performance** (single circuit representation)
  - Circuit is exclusively Boolean or Arithmetic

- Since complexity of 2PC protocol grows with circuit size, for performance,
  - Require Arithmetic circuit for , Boolean circuit for comparison with
- **None** of the high-level frameworks support a mix of Arithmetic and Boolean circuit

- Since complexity of 2PC protocol grows with the circuit size, for performance,
  - Require Arithmetic circuit for $w^t x$, Boolean circuit for comparison with b
- **None** of the high-level frameworks support a mix of Arithmetic and Boolean circuit

Current state of affairs

Function: $w^t x > b$

<u>Option 2</u>

- Program in ABY framework (Demmler et al. NDSS-15)

- Pro: Uses a combination of Boolean and Arithmetic circuits
- Pro: Much better performance

- Cons: **Not programmer friendly** (low level)
  - Manually split compute into Boolean & Arithmetic
  - Write corresponding low-level circuits for each part
  - Insert inter-conversions between them
- Cons: **Tedious and Error-prone + some**

Current state of affairs

Function:

Function: $w^t x > b$

Option 3

- Design *specialized* protocols for functions of interest
- Pro: Good performance
- Cons: Requires a lot of **cryptographic expertise**
- Cons: **No generality**: Great effort for each function

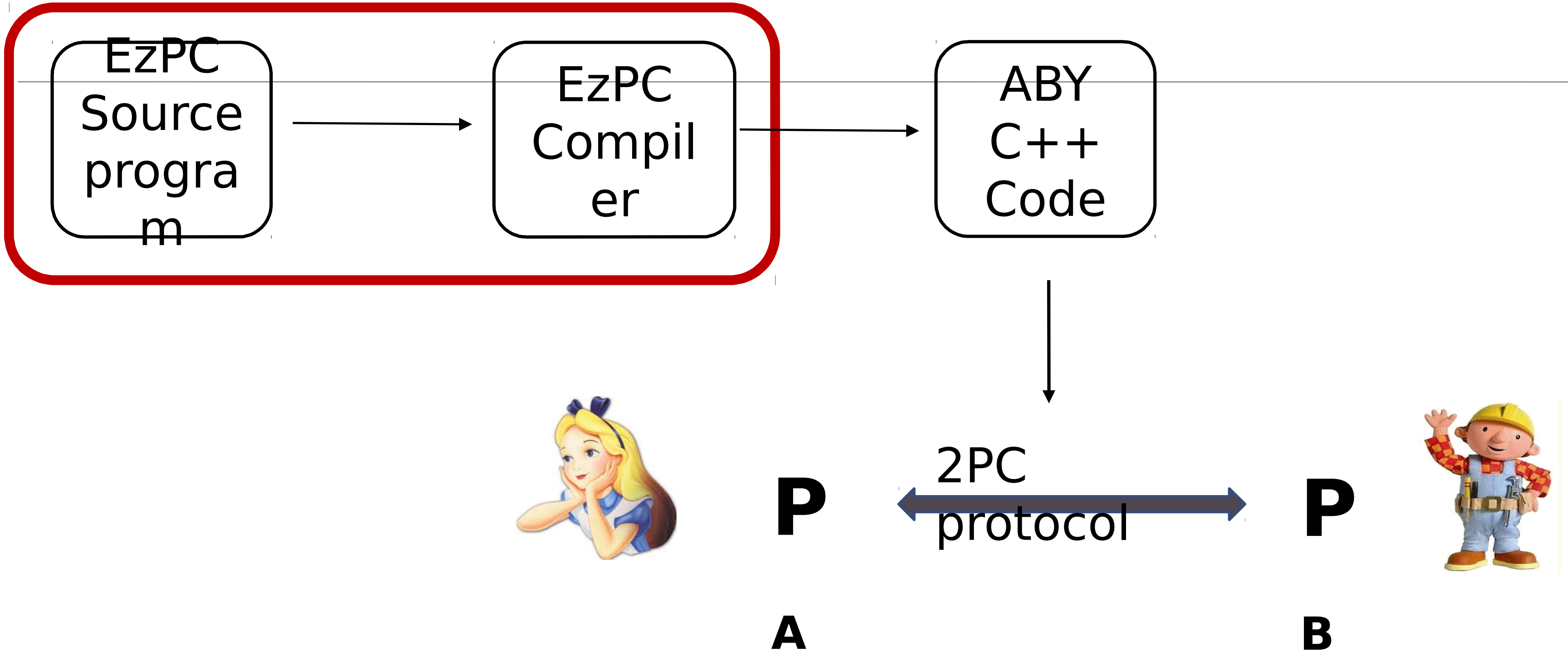
State of the art in 2PC (for $F = (w^T x > b)$)

State of the art in 2PC (for)

Solution	Programmability	Generality	Performance	Scalability	Security
DSLs like ObliVM, CBMC-GC, etc	✔	✔	✗	✔	✔
ABY	✗	✔	✔	✗	✔
Specialized Protocols like MiniONN, etc	✗	✗	✔	✔	✔
EzPC	✔	✔	✔	✔	✔

Our Approach

Our tool: EzPC (Easy two-party computation)



EzPC Source Language → EzPC Compiler → Scalability → Formal Guarantees → Evaluation

EzPC: Source Program

Function: $w^T x > b$

```
uint w[30] = input1();
uint x[30] = input2();
uint b = input1();

uint acc = 0;
for i in [0:30] {
acc = acc + (w[i] * x[i]);
}

Output2(acc > b ? 1 : 0);
```

- Base types and array types
- Mathematical operators (+, *, >, &, >>, ….)
- Statements for assignments, array read/write, bounded for loops and if conditions

```
//circuit builders for arithmetic and boolean
Circuit* ycirc = s[S_YAO]->GetCircuitBuildRoutine();
Circuit* acirc = s[S_ARITH]->GetCircuitBuildRoutine();
...
if(role == SERVER) {
  //Put gates to read w and b
} else { //role == CLIENT
  //Put gates to read x
}

for(uint32_t i = 0; i < 30; i++) { //acc = w^T x
  share * a_t_0 = acirc->PutMULGate(a_w[i], a_x[i]);
  a_acc = acirc->PutADDGate(a_acc, a_t_0 );
}

//convert acc and b from arithmetic to boolean
share *y_acc = ycirc->PutA2YGate(a_acc);
share *y_b = ycirc->PutA2YGate(a_b);

share *y_pred = ycirc->PutGTGate (y_acc, y_b);
uint32_t one = 1 ;
share *y_1 = ycirc->PutCONSGate(one, bitlen);
uint32_t zero = 0 ;
share *y_0 = ycirc->PutCONSGate(zero, bitlen);
share *y_t = ycirc->PutMUXGate(y_pred, y_1, y_0);

share *y_out = ycirc->PutOUTGate(y_t, CLIENT);
party->ExecCircuit();

if(role==CLIENT) { //only to the client
  uint32_t _o = y_out->get_clear_value<uint32_t>();
}
```

providing low-level circuit API

EzPC: How the compiler works?

- EzPC source program → ABY code
- Problem: Automatically assigns variables and operators to **B**oolean or **A**rithmetic type
- Using cryptographic costs of primitive operators as well as inter-conversion costs

- Hard problem, can require exponential time
- Heuristics-based cryptographic cost-aware compiler

Our heuristics

- Hard Constraints: MULT in Arithmetic; GT/COND/BitwiseAND in Boolean
- Soft Constraints: ADD can be either in Arithmetic or Boolean based on operands

- Minimize inter-conversion cost
  - Maintain a map from variables to available types

EzPC: Cryptographic Cost-aware Compiler

- Hard Constraints: MULT in Arithmetic; GT/COND/BitwiseAND in Boolean
- Soft Constraints: ADD can be either in Arithmetic or Boolean based on operands

```
uint w[30] = input1();
uint x[30] = input2();
uint b = input1();

uint acc = 0;
for i in [0:30] {
uint temp = w[i] * x[i];
acc = acc + temp;
}

Output2(acc > b ? 1 : 0);
```

Source Program

```
uint^A w[30] = input1();
uint^A x[30] = input2();
uint^B b = input1();

uint^A acc = 0;
for i in [0:30] {
uint^A temp = w[i] *_A x[i];
acc = acc +_A temp;
}

uint^B acc_B = A2B(acc);
Output2(acc_B >_B b ?_B 1 : 0);
```

Intermediate Program
(Annotate all variables & operators and insert inter-conversions)

EzPC: Cryptographic Cost-aware Compiler

– Minimize inter-conversions: Maintain a map from variables to available share type

```
⋮
uint acc = 0;
for i in [0:30] {
uint temp = w[i] * x[i];
acc = acc + temp;
}

uint o = (acc > b ? 1 : 0);
uint y = acc * w[0];
uint z = acc & b;
```

Source Program

```
⋮
uint^A acc = 0;
for i in [0:30] {
uint^A temp = w[i] *_A x[i];
acc = acc +_A temp;
}

uint^B acc_B = A2B(acc);
uint^B o = (acc_B >_B b ?_B 1 : 0);

uint^A y = acc *_A w[0];
uint^B z = acc_B &_B b;
```

Intermediate Program

EzPC: Scalability

- Program needs to be written as circuit
  - Circuit needs to fit in memory
- Unroll the loops (Circuits don't have loops)
  - Circuit size can be huge (>28 GB)
- Secure Code Partitioning
  - Partition into P1 and P2
  - Need to pass Acc to P2 *securely*
  - Secret-share Acc b/w Alice & Bob
  - P2 reconstructs Acc
- Very natural and crucial for benchmarks such as large DNNs,

w, u

x, v

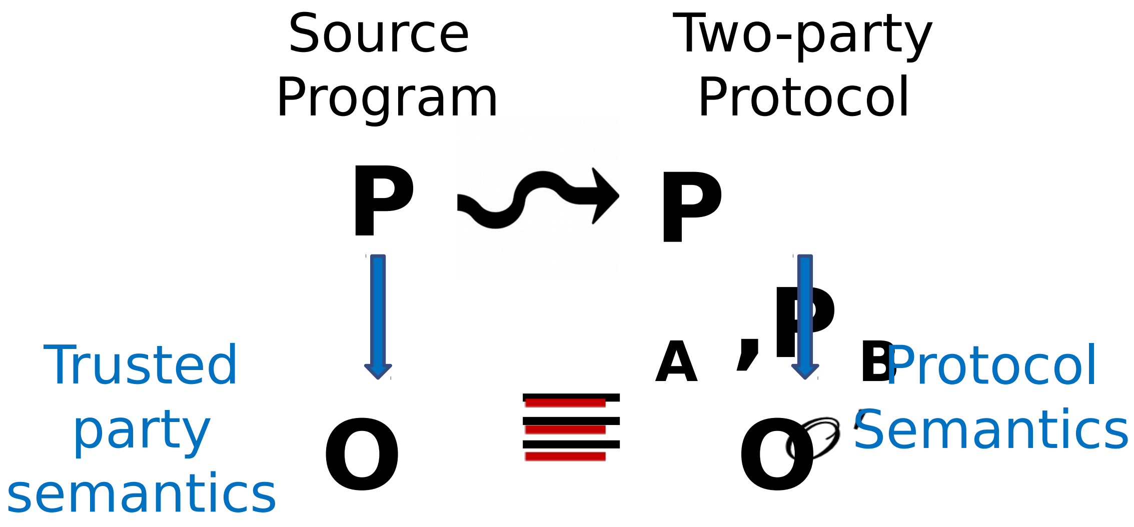
$s_A := R_A$

$s_B :=$ Acc - R_A

Revealing Acc Alice or Bob breaks security

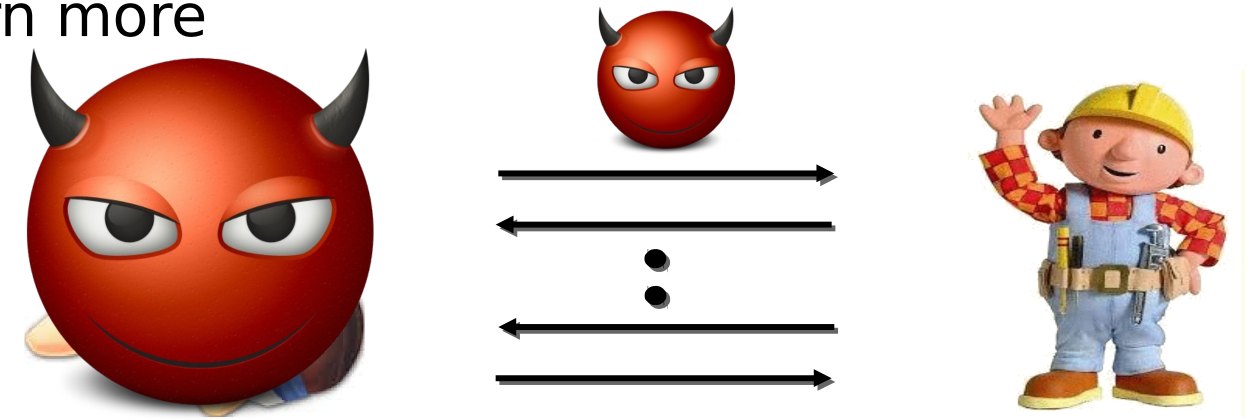
EzPC: Formal Guarantees

- **Correctness**
  - Formulate trusted party semantics and protocol semantics
  - For a well-typed **P,** both semantics
    - terminate without errors
    - produce same outputs
  - No array index out of bounds errors



EzPC: Formal Guarantees

- **Security**
  - Semi-honest security against corruption of one party
  - Honest-but-curious adversary that follows the protocol faithfully BUT is eager to learn more



- Eavesdrop on communication
  - Learns nothing about Alice’s or Bob’s input
- Corrupt Alice
  - Learn nothing about Bob’s input (beyond o/p)

- Formally reduce security of compiler to semi-honest security of 2PC back-end (ABY)
- Security of partitioning scheme

Security of 2PC back-end → Security of EzPC programs

Applications of EzPC to Private Machine Learning

Secure Prediction using Secure 2PC

F(Model, Data)

2PC protocol

ML classifier for diabetes

Model is IP

Medical report

Data is private

- Bob wants to learn output of classifier
- Solved by 2PC!
- Bob learns classifier output *only*
- Azure learns *nothing* about Bob's input!

EzPC: Evaluation

- Demonstrate generality by evaluating EzPC on large variety of benchmarks
- In *all* cases, EzPC protocols BEAT/MATCH performance of state-of-the-art specialized protocols
- Writing benchmarks did not require any crypto know-how
- Lines of code (LOC) is proportional to C++ code for describing the ality

Generic 2PC protocols gives state-of-art performance (if done smartly)!

Deep Neural Networks

- Many layers; Each layer has
  - A linear operation that can be written as a matrix multiplication
  - A non-linear activation function such as Maxpool, ReLU, etc
- Matrix multiplication is suited for Arithmetic; non-linear function is suited to Boolean

- **Cryptonets** (ICML 16)
  - Based on Homomorphic Encryption (HE)
  - MNIST: 1 fully connected, 1 convolutional, square activation function

DNN	Cryptonets Time (s)	EzPC Time (s)
Cryptonets	297	0.6

2PC based approach much faster than HE!

Our Evaluation

Benchmark	Prev. Time (s)
Naïve Bayes (Audiology)	3.9
Decision Trees (ECG)	0.4
SecureML (MNIST)	1.1
MiniONN (MNIST)	9.4
MiniONN CIFAR-10	544

More ML classifiers in EzPC

- Tensorflow tutorial benchmarks
  - Softmax regression for MNIST: $\arg\max W \cdot x + b$
  - DNN for MNIST: 2 convolutional, 2 fully connected, ReLU activation, 99.2% accuracy

- Tensorflow tutorial benchmarks
  - Softmax regression for MNIST: $\arg\max W \cdot x + b$
  - DNN for MNIST: 2 convolutional, 2 fully connected, ReLU activation, 99.2% accu

	LAN (1ms) Time (s)	WAN (40ms) Time (s)	LOC
Regression	0.1	0.7	38
DNN	30.5	60.3	172

First 2PC implementations for these benchmarks

- Bonsai (ICML 17): Much smaller models for weak IoT devices, reasonable accuracy
  - Tree like structure of the model

- Bonsai (ICML 17): Much smaller models for weak IoT devices, reasonable accuracy
  - Tree

Dataset	Depth	LAN (1ms) Time (s)	WAN (40ms) Time (s)	LOC
USPS	2	0.2	0.9	156
WARD	3	0.3	1.1	283

Demonstrates programmability and generality of EzPC

EzPC: In a nut-shell

- Developer friendly
  - Easy to get correct functionality

- Generality and Customizability
  - Small change in functionality requires small change in code

- State-of-the-art performance
  - Beats specialized protocols

- Scales to large programs

- Formal guarantees of correctness and security

Future Directions

- Generalize EzPC to more than 2 parties
  - Integrate existing MPC protocols to EzPC
  - Build new MPC protocols that combine Arithmetic and Boolean
- Malicious parties?
- Make language of EzPC more powerful
  - Enhance the expressiveness of the language with functions
  - Better support for floating point operations
- Find other exciting applications apart from private machine learning