Preserving Privacy through Processing Encrypted Data

Prof. Miriam Leeser

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA
mel@coe.neu.edu

Joint work with Prof. Stratis Ioannidis

Analyzing Data from Human Subjects

- Long history in experimental/life sciences
  - Medicine
  - Psychology
  - Sociology
  - Behavioral Economics
  - …

- Ubiquitous practice
  - Display & search ads
  - e-Commerce
  - Streaming
  - ...

- Integral to daily operations
  - User profiling
  - Targeted advertising
  - Personalized recommendations

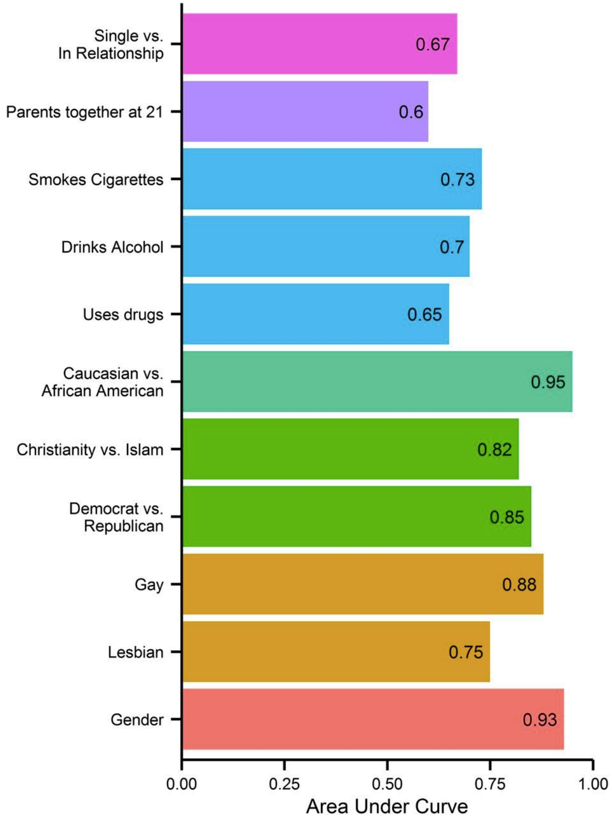
Privacy threat exists because

…personal information can be inferred from seemingly innocuous behavioral data

Observed in several contexts:

- ❑**Facebook likes** [Kosinski, Stillwell, Graepel, 2013]
  - ➢ Gender, Political Affiliation, Age, Religion, Marital Status, Smoking, Sexual Orientation, Drug Use.
- ❑**Search queries** [Bi, Shokouhi, Kosinski, Graepel, 2013].
  - ➢ Gender, Political Affiliation, Age, Religion
- ❑**Tweets** [Rao, Yarowsky, Shreevats, Gupta, 2010]
  - ➢ Gender, Political Affiliation, Age, Origin.
- ❑**Movie Ratings** [Weinsberg, Bhagat,Ioannidis,Taft, 2012]
  - ➢ Gender, Political Affiliation, Age

[Kosinski, Stillwell, Graepel, 2013]

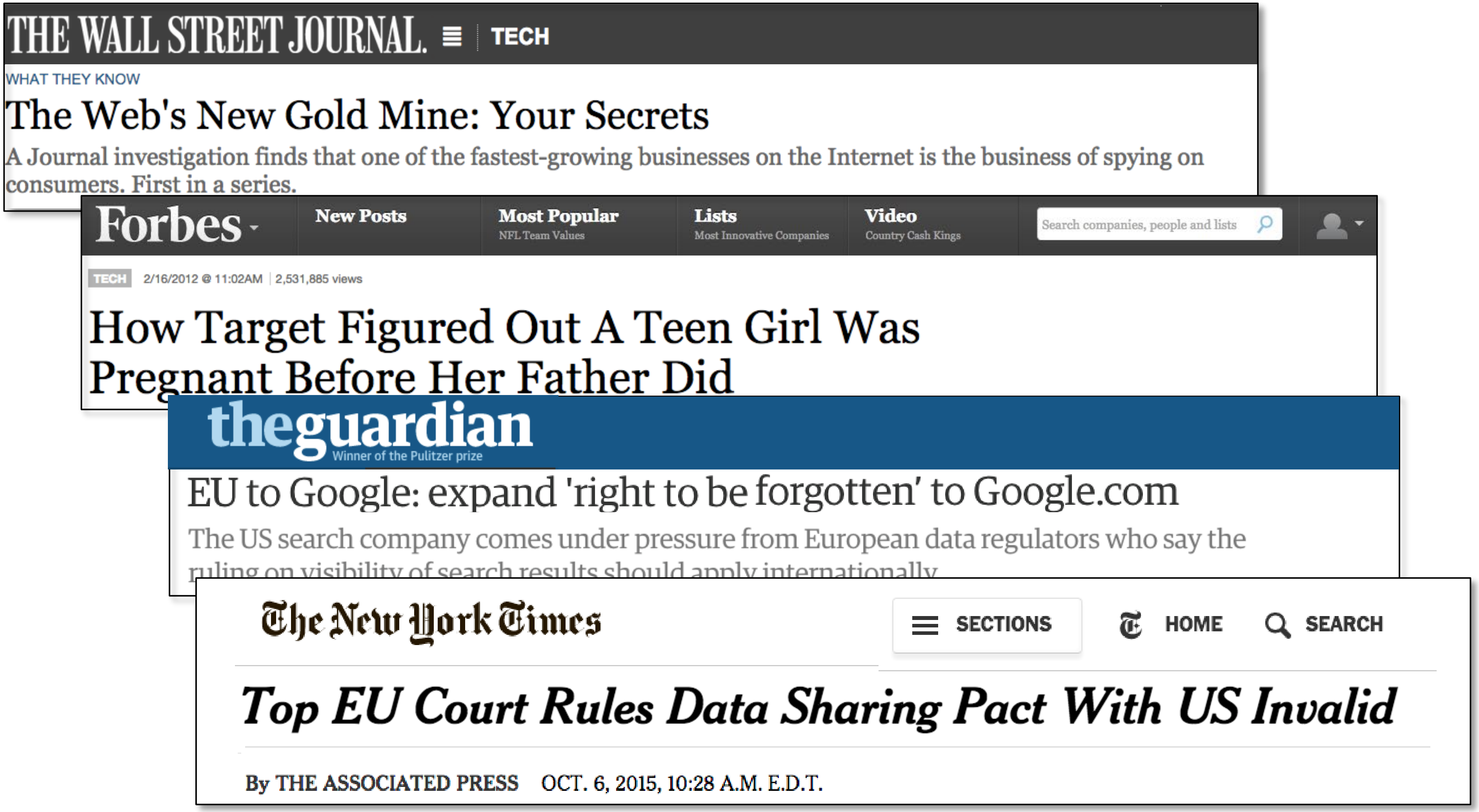


Privacy threat exists because…

…information is coming from an increasingly diverse pool of sources

- □ Implicit (e.g., queries, clicks, purchases, views)
- □ Explicit (e.g., ratings, likes)
- □ Mobile devices
- □ Integrated/embedded sensors
- □ …

Privacy Concerns Well Documented



WHAT THEY KNOW

The Web's New Gold Mine: Your Secrets

A Journal investigation finds that one of the fastest-growing businesses on the Internet is the business of spying on consumers. First in a series.

TECH 2/16/2012 @ 11:02AM | 2,531,885 views

How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did

EU to Google: expand 'right to be forgotten' to Google.com

The US search company comes under pressure from European data regulators who say the ruling on visibility of search results should apply internationally

Top EU Court Rules Data Sharing Pact With US Invalid

By THE ASSOCIATED PRESS OCT. 6, 2015, 10:28 A.M. E.D.T.

Motivating Question

❑Q: Can we enable the mining and analysis of sensitive datasets, while offering privacy guarantees?

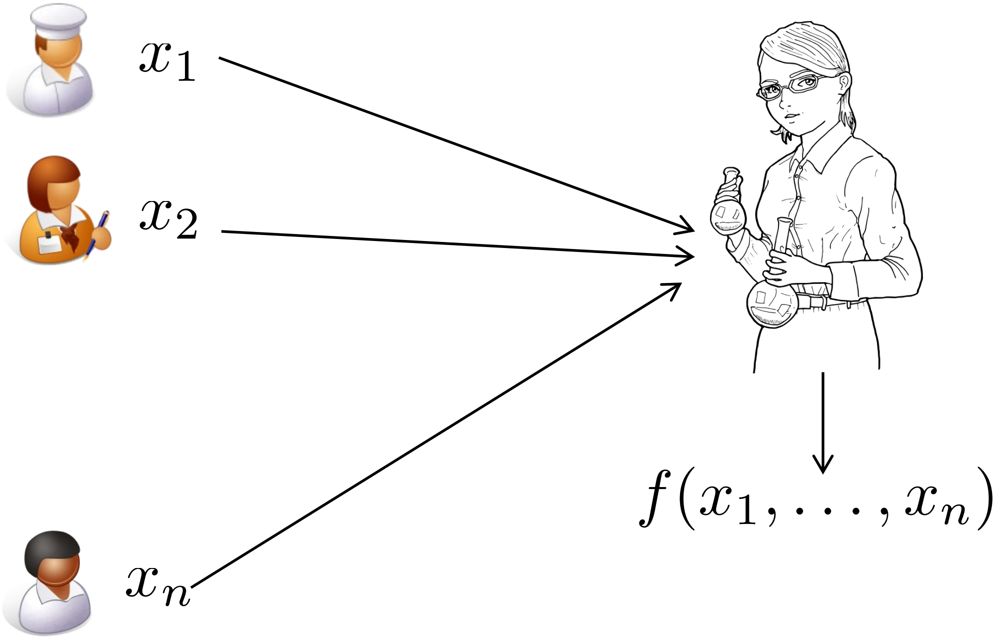
❑A: Yes, through SFE:

Secure Function Evaluation

SFE: Canonical Problem: Millionaires

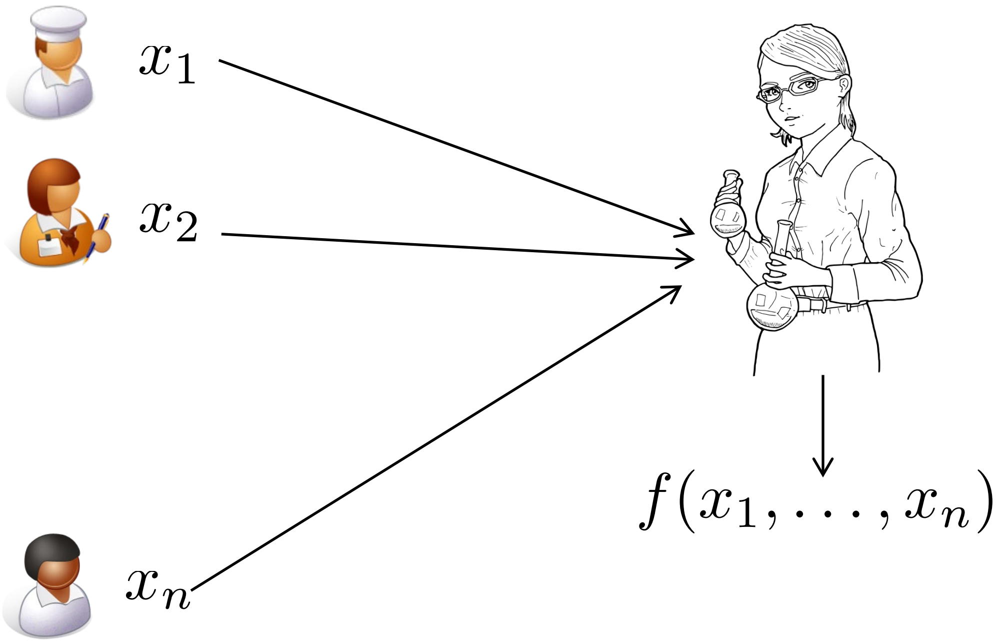
- Current state of the art
  - Users transmit encrypted data
  - Evaluator decrypts data for processing
    - Evaluator or malicious third party could access decrypted data
- SFE: Secure Function Evaluation
  - Evaluator processes encrypted data
  - In our model, everyone knows function
    - Only user has access to his/her unencrypted data

Secure Function Evaluation (SFE)



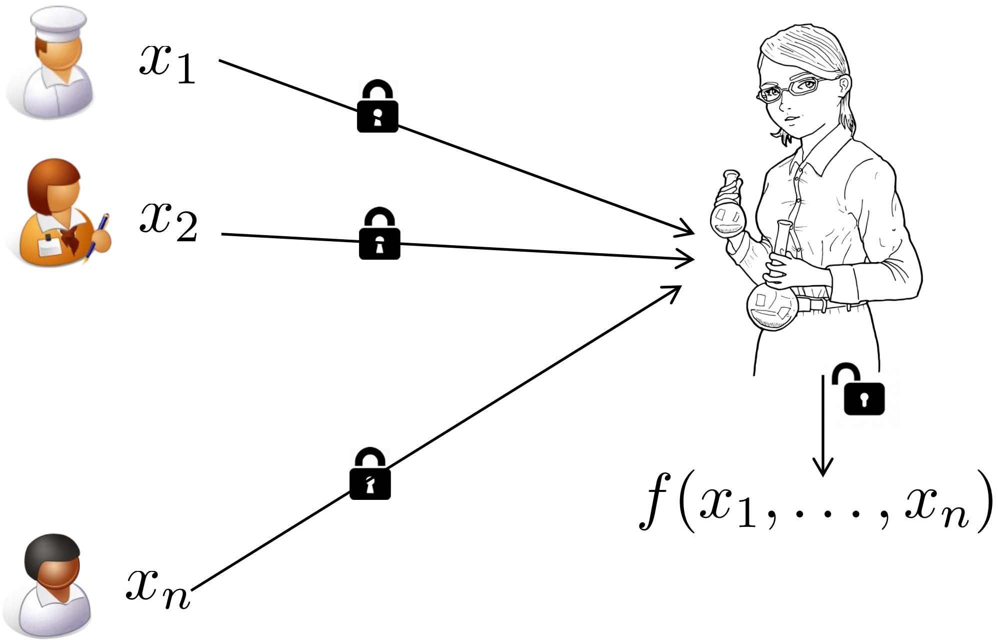
The analyst **learns only** $f(x_1, \ldots, x_n)$**, and nothing else.**

Grand Challenge



Apply SFE to execute **real-life**, **practical algorithms** over **massive datasets**

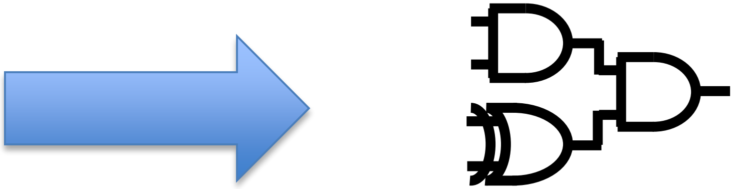
Challenge 1: Overhead Due to Encryptions



Slow-down can be of several orders of magnitude (e.g., 100,000 times slower).

Challenge 2: Cost of Obliviousness

```
function bs(val, s, t)
  mid = (s + t) / 2;
  if (val < mem[mid])
    bs(val, 0, mid)
  else
    bs(val, mid+1, t)
```

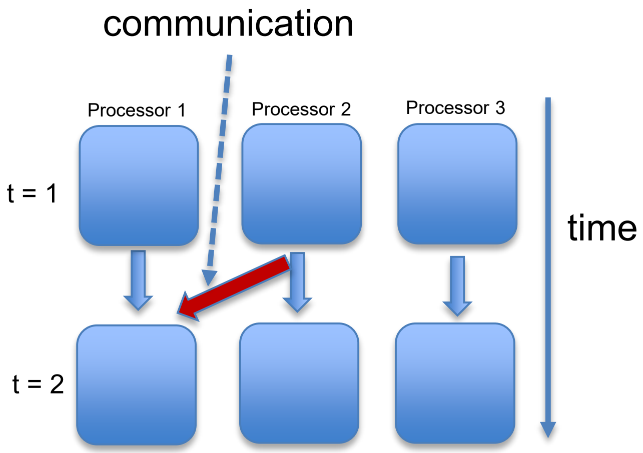


□ Translating to **data-oblivious algorithm** can **increase total work**

□ For “big data”, even going from $O(n)$ to $O(n^2)$ is prohibitive

Challenge 3: Maintaining Parallelism

- ❑ Desirable properties:
  - ❑Low parallel processing time
  - ❑Low communication cost
- ❑ Very important for “big data”



Caveat: Communication **patterns** between processors ***may reveal something about the data***

Our Contributions (Ioannidis and Leeser)

- ❑ Generic approach for SFE of **graph-parallel algorithm**
  - ❑ **Scatter/Gather/Apply**
  - ❑ Includes several important DM+ML algorithms
    - ❑ Matrix/Tensor Factorization, Training DNNs, …

- ❑ GraphSC: Highly parallel implementation
  - – Total Work: $O(M\log^2 M)$ Blowup: $O(\log^2 M)$
  - – Parallel Time: $O(\log^2 M)$
  - – Generic Implementation -- no accelerators
  - – MF: 1M ratings, 128 cores: 13 hours

- ❑ Hardware implementation & acceleration via FPGAs in the Datacenter
  - ❑ First published in FPGA 2017

Overview

❑Background: Yao's Garbled Circuits

❑Acceleration via FPGAs

Basic Ingredient: Proxy Oblivious Transfer

chooser

receiver

sender

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

$m_0 \, , m_1$

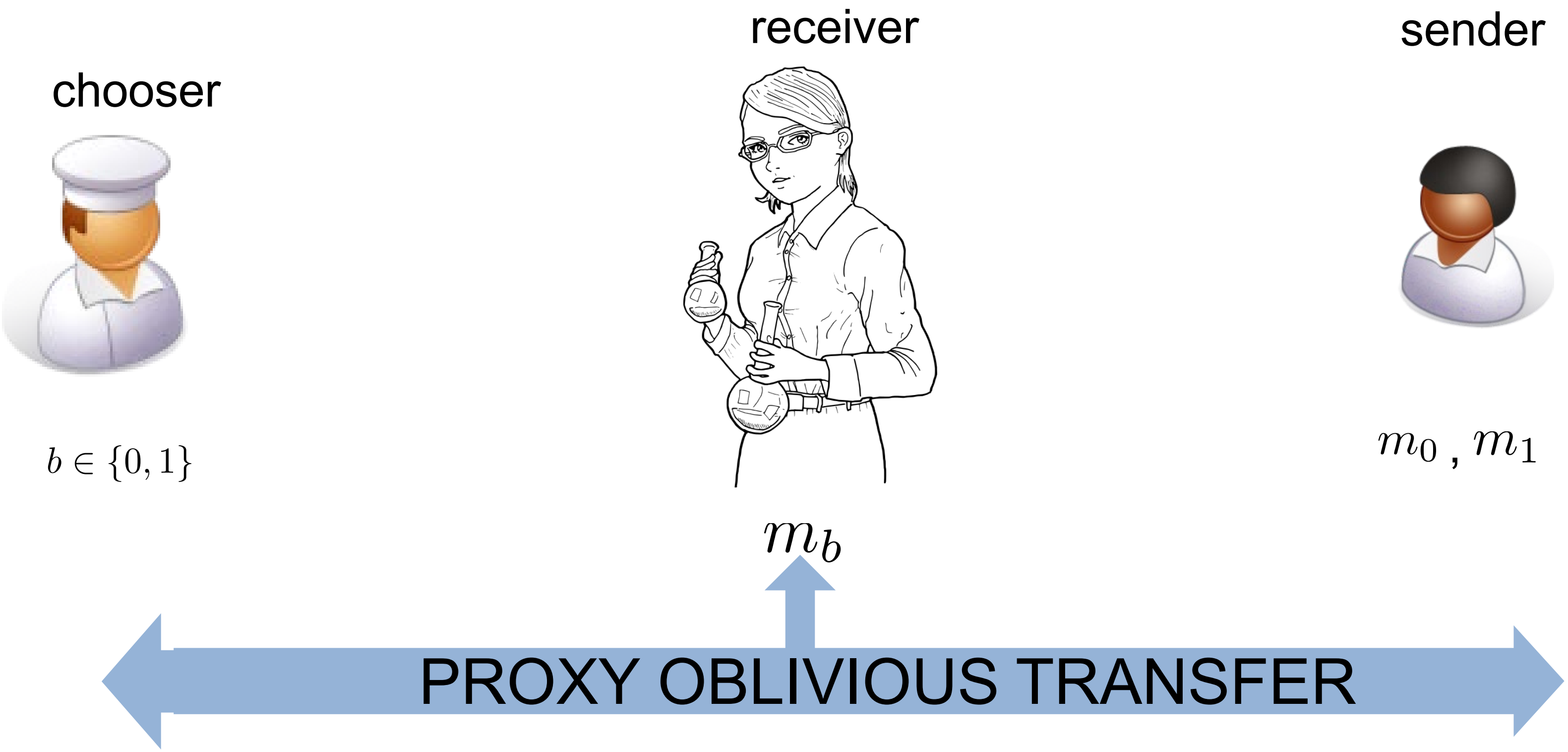
Initially:

□ Chooser has a **secret bit**

□ Sender has two **secret messages**

□ Receiver knows nothing

Basic Ingredient: Proxy Oblivious Transfer



chooser

receiver

sender

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

m_b

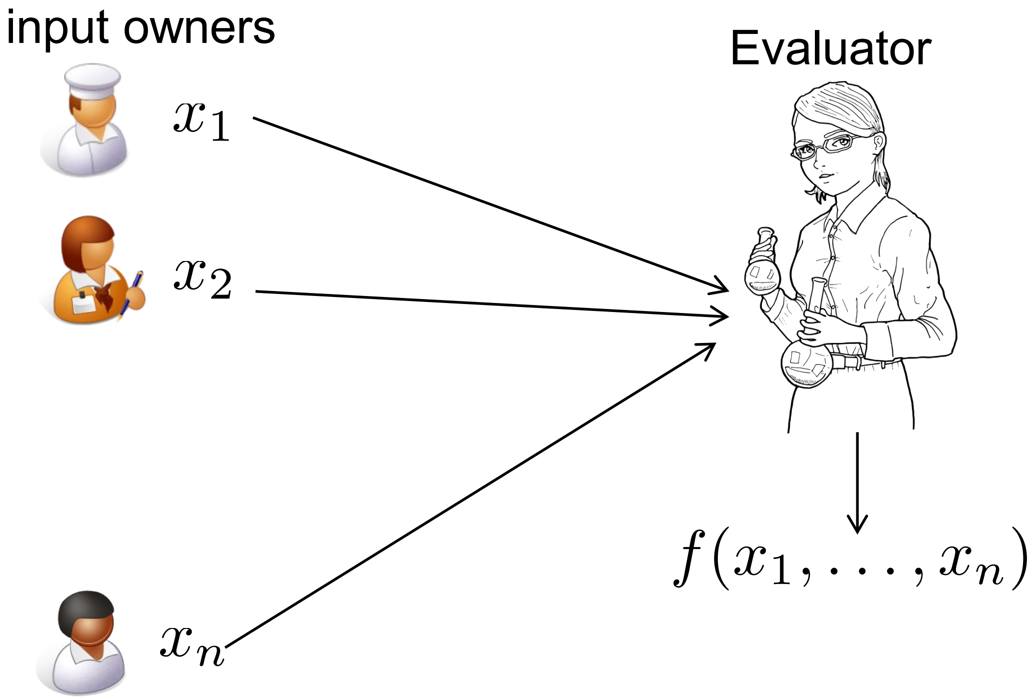
m_0 , m_1

PROXY OBLIVIOUS TRANSFER

At the conclusion of the protocol:

- ❑ Receiver learns **secret message corresponding to secret bit** (i.e., m_b)
- ❑ Receiver learns neither b nor $m_{\bar{b}}$
- ❑ Sender and chooser learn nothing

Yao's Garbled Circuits (GC)



The evaluator **learns *f(…)* and nothing but *f(…)***

❑ GC requires expressing f as a Boolean Circuit

Yao’s Garbled Circuits Example: AND gate

input owners

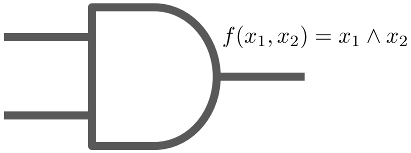
x_1

x_2

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1



The evaluator **learns *f(…)* and nothing but *f(…)***

The garbler facilitates SFE but **learns nothing**

Yao's Garbled Circuits

input owners

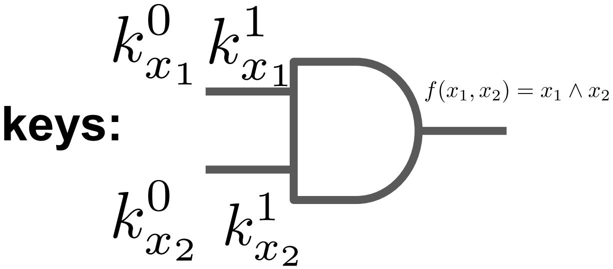
x_1

x_2

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1



For each **input** wire, the garbler generates 2 **random strings** (representing 0 and 1, respectively)

Yao’s Garbled Circuits

input owners

x_1

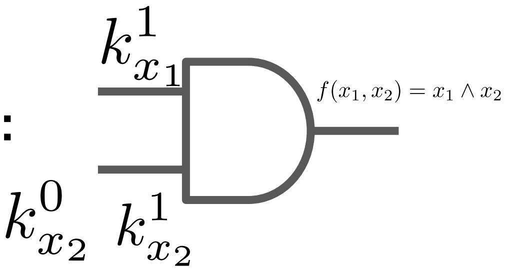
x_2

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k^0_{x_1}$	0	0
$k^0_{x_1}$	1	0
1	0	0
1	1	1

keys:



For each **input** wire, the garbler generates 2 **random strings** (representing 0 and 1, respectively)

Yao's Garbled Circuits

input owners

x_1

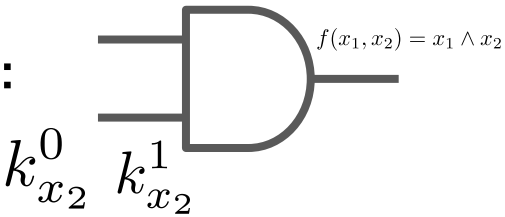
x_2

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	0	0
$k_{x_1}^0$	1	0
$k_{x_1}^1$	0	0
$k_{x_1}^1$	1	1

keys:



For each **input** wire, the garbler generates 2 **random strings** (representing 0 and 1, respectively)

Yao's Garbled Circuits

input owners

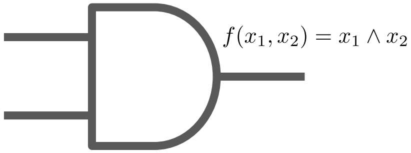
x_1

x_2

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	$k_{x_2}^0$	0
$k_{x_1}^0$	$k_{x_2}^1$	0
$k_{x_1}^1$	$k_{x_2}^0$	0
$k_{x_1}^1$	$k_{x_2}^1$	1



For each **input** wire, the garbler generates 2 **random strings** (representing 0 and 1, respectively)

Yao’s Garbled Circuits

input owners

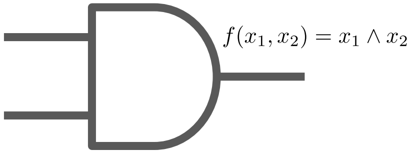
x_1

x_2

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	$k_{x_2}^0$	$E_{k_{x_1}^0, k_{x_2}^0}[0]$
$k_{x_1}^0$	$k_{x_2}^1$	$E_{k_{x_1}^0, k_{x_2}^1}[0]$
$k_{x_1}^1$	$k_{x_2}^0$	$E_{k_{x_1}^1, k_{x_2}^0}[0]$
$k_{x_1}^1$	$k_{x_2}^1$	$E_{k_{x_1}^1, k_{x_2}^1}[1]$



$$E_{k,k'}[m] = \texttt{SHA1}(k \| k') \oplus m$$

The garbler **encrypts** each of the 4 possible outputs using the **corresponding pairs input keys**

Yao’s Garbled Circuits

input owners

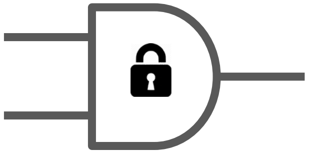
x_1

x_2

Evaluator

Garbler

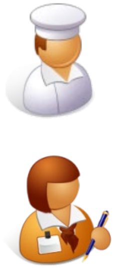
Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	$k_{x_2}^0$	$E_{k_{x_1}^0, k_{x_2}^0}[0]$
$k_{x_1}^0$	$k_{x_2}^1$	$E_{k_{x_1}^0, k_{x_2}^1}[0]$
$k_{x_1}^1$	$k_{x_2}^0$	$E_{k_{x_1}^1, k_{x_2}^0}[0]$
$k_{x_1}^1$	$k_{x_2}^1$	$E_{k_{x_1}^1, k_{x_2}^1}[1]$



Garbled gate: 4 encrypted ciphertexts, **permuted**.

Yao’s Garbled Circuits

input owners



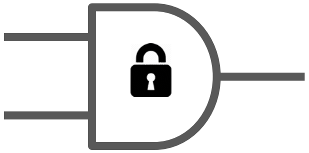
x_1

x_2

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k^0_{x_1}$	$k^0_{x_2}$	c_1
$k^0_{x_1}$	$k^1_{x_2}$	c_2
$k^1_{x_1}$	$k^0_{x_2}$	c_3
$k^1_{x_1}$	$k^1_{x_2}$	c_4



Garbled gate: 4 encrypted ciphertexts, **permuted**.

Yao’s Garbled Circuits

input owners

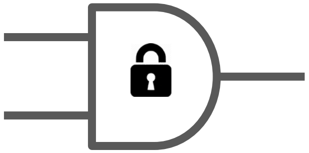
x_1

x_2

Evaluator

Garbler

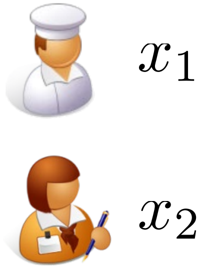
Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	$k_{x_2}^0$	c_1
$k_{x_1}^0$	$k_{x_2}^1$	c_2
$k_{x_1}^1$	$k_{x_2}^0$	c_3
$k_{x_1}^1$	$k_{x_2}^1$	c_4



Garbled gate: 4 encrypted ciphertexts, **permuted**.

Yao's Garbled Circuits

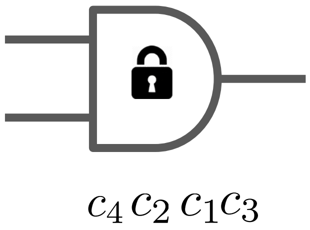
input owners



Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	$k_{x_2}^0$	c_1
$k_{x_1}^0$	$k_{x_2}^1$	c_2
$k_{x_1}^1$	$k_{x_2}^0$	c_3
$k_{x_1}^1$	$k_{x_2}^1$	c_4



Garbler sends garbled gate to **Evaluator.**

Yao’s Garbled Circuits

input owners

x_1

x_2

Evaluator

Garbler

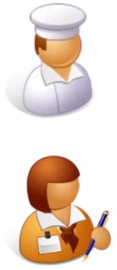
Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	$k_{x_2}^0$	c_1
$k_{x_1}^0$	$k_{x_2}^1$	c_2
$k_{x_1}^1$	$k_{x_2}^0$	c_3
$k_{x_1}^1$	$k_{x_2}^1$	c_4

$c_4\, c_2\, c_1 c_3$

Garbler, Evaluator, and input owners engage in **proxy oblivious transfer.**

Yao’s Garbled Circuits

input owners



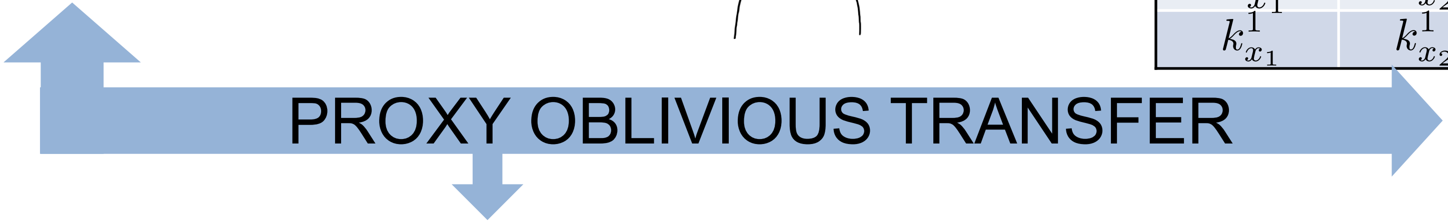
$x_1 = 0$

$x_2 = 1$

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k^0_{x_1}$	$k^0_{x_2}$	c_1
$k^0_{x_1}$	$k^1_{x_2}$	c_2
$k^1_{x_1}$	$k^0_{x_2}$	c_3
$k^1_{x_1}$	$k^1_{x_2}$	c_4



PROXY OBLIVIOUS TRANSFER

$k^0_{x_1}, k^1_{x_2}$

$c_4\, c_2\, c_1 c_3$

Garbler, Evaluator, and input owners engage in **proxy oblivious transfer.**

As a result, **Evaluator** learns keys corresponding to true inputs, while **Garbler** learns nothing.

Yao’s Garbled Circuits

input owners

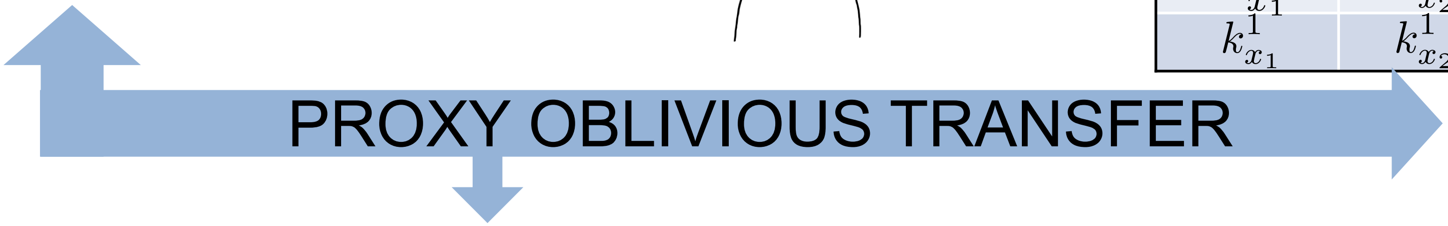
$x_1 = 0$

$x_2 = 1$

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k_{x_1}^0$	$k_{x_2}^0$	c_1
$k_{x_1}^0$	$k_{x_2}^1$	c_2
$k_{x_1}^1$	$k_{x_2}^0$	c_3
$k_{x_1}^1$	$k_{x_2}^1$	c_4



PROXY OBLIVIOUS TRANSFER

$k_{x_1}^0, k_{x_2}^1$

$c_4 \, c_2 \, c_1 c_3$

$$D_{k,k'}[m] = \texttt{SHA1}(k \| k') \oplus m$$

Evaluator attempts to decrypt each of the four ciphertexts; the decryption that succeeds reveals the **true output.**

Yao's Garbled Circuits

input owners

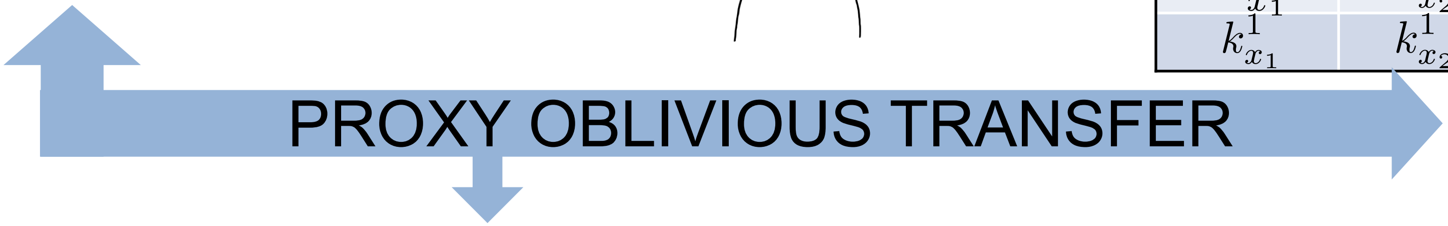
$x_1 = 0$

$x_2 = 1$

Evaluator

Garbler

Inputs		Outputs
x_1	x_2	$x_1 \wedge x_2$
$k^0_{x_1}$	$k^0_{x_2}$	c_1
$k^0_{x_1}$	$k^1_{x_2}$	c_2
$k^1_{x_1}$	$k^0_{x_2}$	c_3
$k^1_{x_1}$	$k^1_{x_2}$	c_4



PROXY OBLIVIOUS TRANSFER

$k^0_{x_1}, k^1_{x_2}$

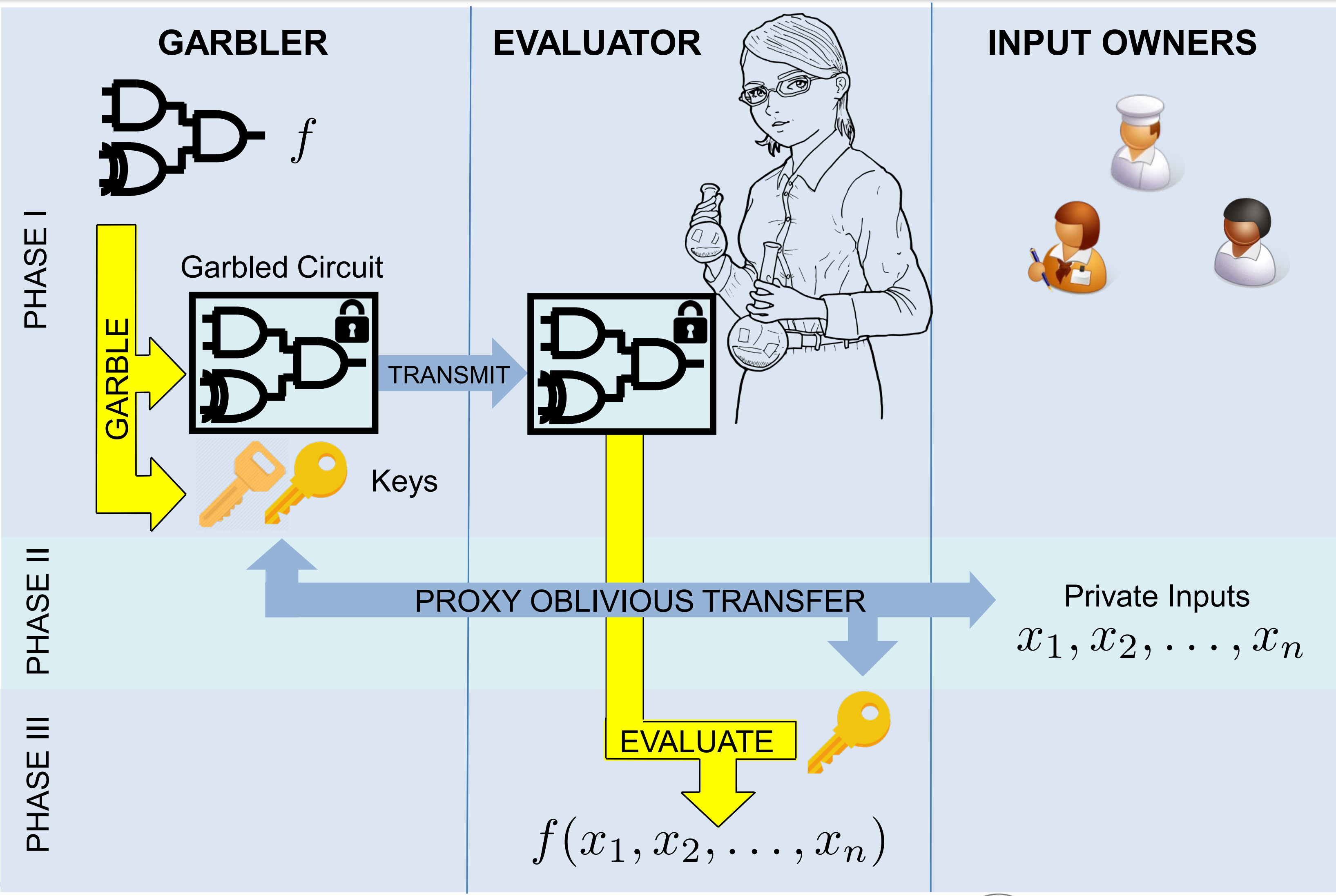
$$D_{k,k'}[m] = \mathtt{SHA1}(k \| k') \oplus m$$

$c_4 \; c_2 \; c_1 c_3$

$$0 = f(x_1, x_2) = x_1 \wedge x_2$$

Evaluator attempts to decrypt each of the four ciphertexts; the decryption that succeeds reveals the **true output.**

Yao's Garbled Circuits: Protocol Overview



GARBLER

EVALUATOR

INPUT OWNERS

PHASE I

f

GARBLE

Garbled Circuit

TRANSMIT

Keys

PHASE II

PROXY OBLIVIOUS TRANSFER

Private Inputs

$x_1, x_2, \ldots, x_n$

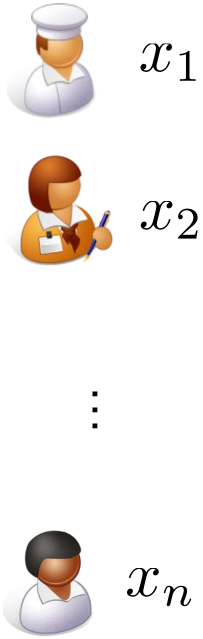
PHASE III

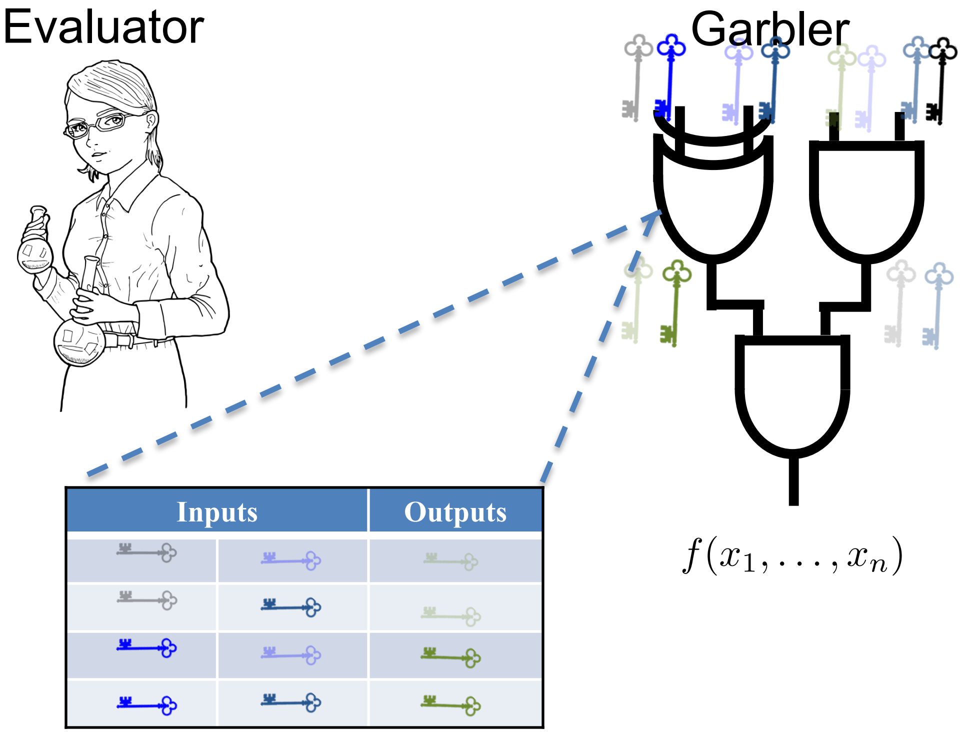
EVALUATE

$f(x_1, x_2, \ldots, x_n)$

Yao’s Garbled Circuits

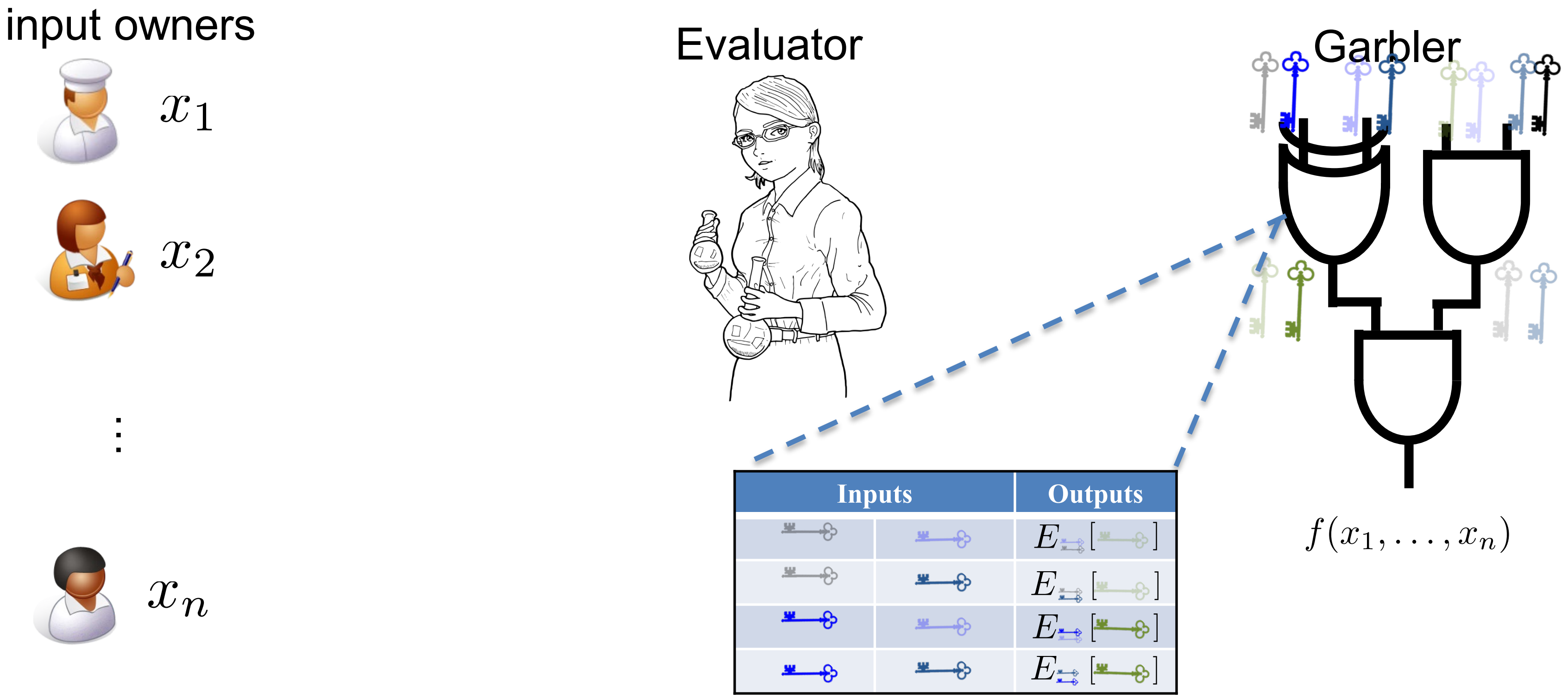
input owners





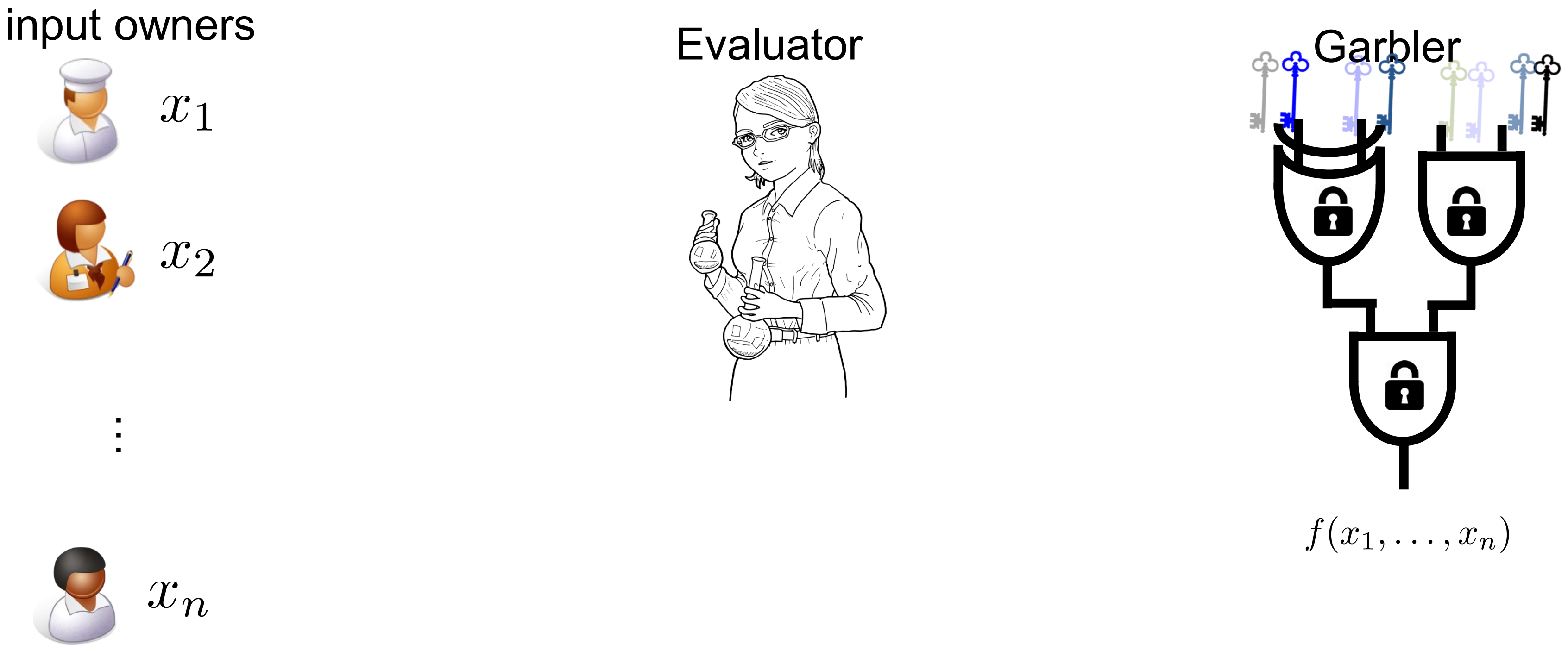
For each **gate**, **input keys** are used to encrypt **output keys**

Yao’s Garbled Circuits



For each **gate**, **input keys** are used to encrypt **output keys**

Yao’s Garbled Circuits



input owners

x_1

x_2

$\vdots$

x_n

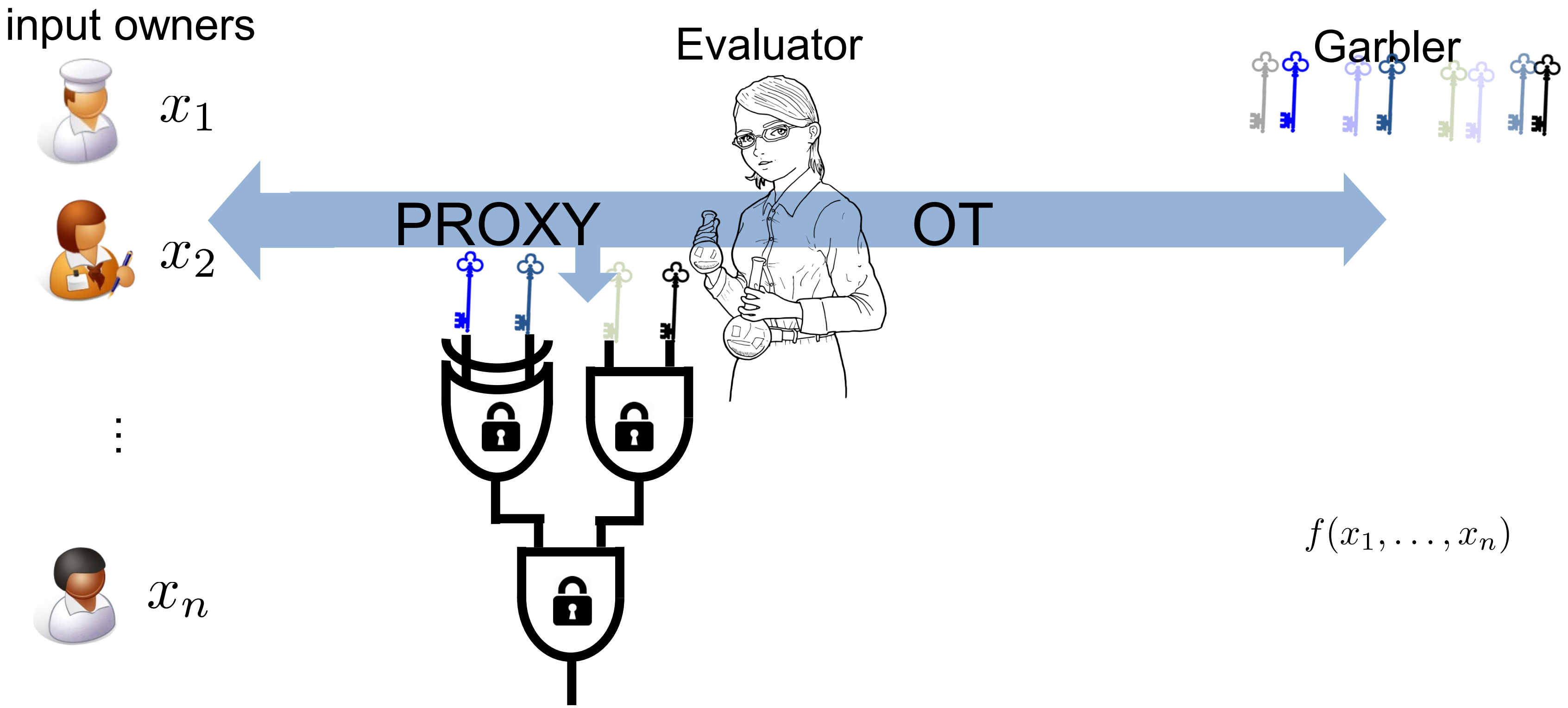
Evaluator

Garbler

$f(x_1, \ldots, x_n)$

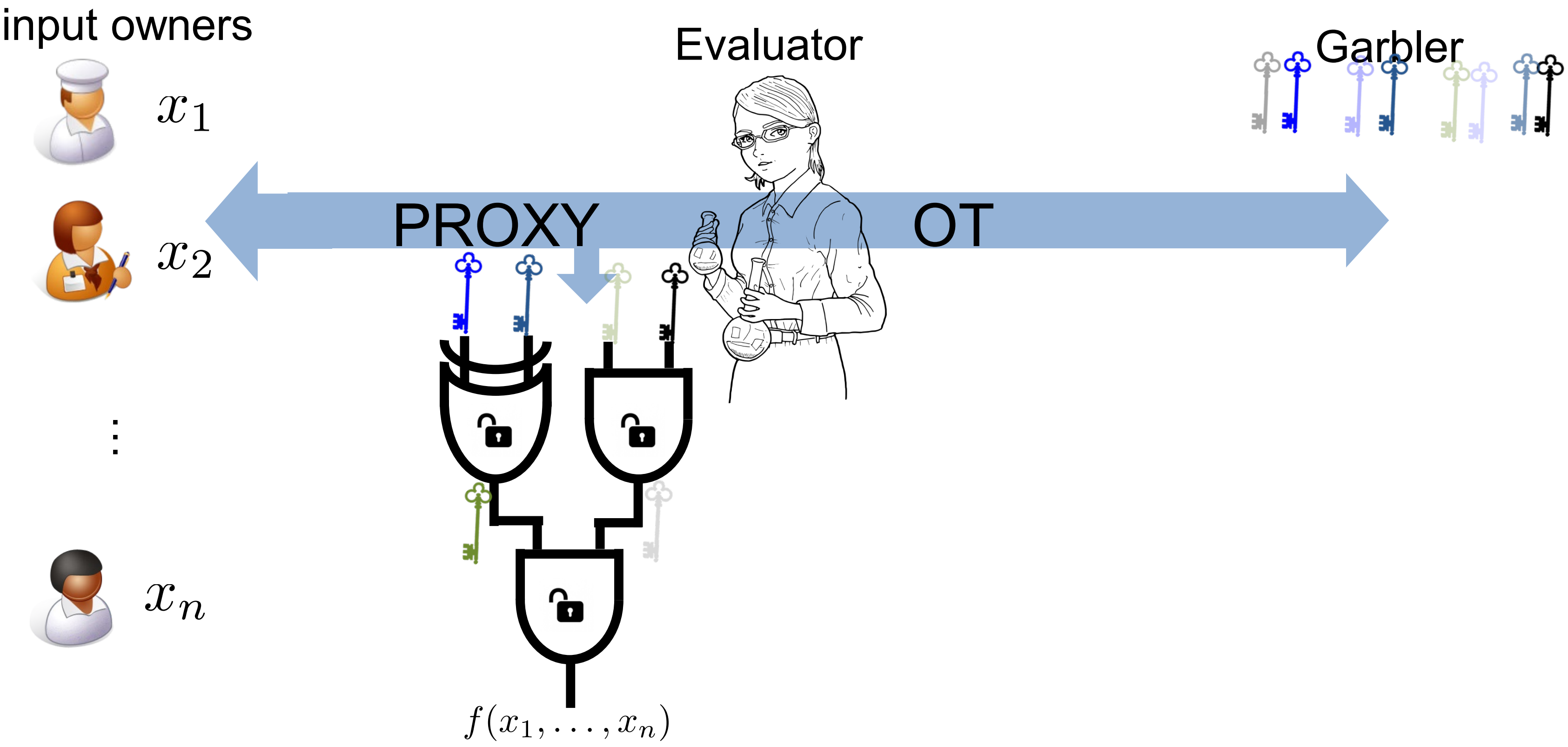
Garbler keeps only input keys, and sends all garbled gates to **Evaluator**

Yao’s Garbled Circuits



Through proxy OT, **Evaluator** learns **keys** corresponding to **true** inputs

Yao’s Garbled Circuits



Using input keys, **Evaluator** “ungarbles” gates, learning final output.

Yao’s Garbled Circuits

GC Protocol Improvements

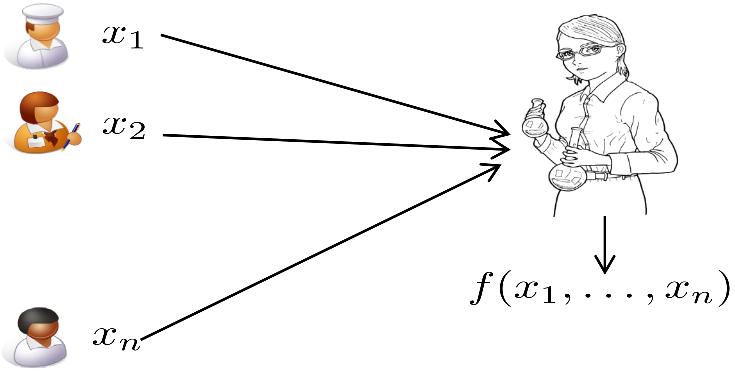
❑ **Row reduction**	[Naor, Pinkas, Summer; EC 1999]
❑ **Point-and-permute**	[Malkhi, Nisan, Pinkas, Sella; USENIX Security 2004]
❑ **Free-XOR**	[Kolesnikov, Schneider; ICALP 2008]
❑ Two halves AND	[Zahur, Rosulek, Evans; EUROCRYPT 2015]

GC Implementation Frameworks

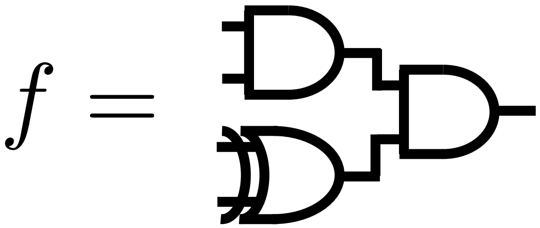
❑ Fairplay	[Malkhi, Nisan, Pinkas, Sella; Usenix 2004]
❑ TASTY	[Henecka, Kogl, Sadeghi, Schneider Wehrenberg; CCS 2010]
❑ FastGC	[Huang, Evans, Katz, Malka; USENIX Security 2011]
❑ **ObliVM**	[Liu, Wang, Nayak, Huang, Shi; IEEE S&P, 2015]
❑ …	

The “SFE for Privacy” Program

- □ Pick data mining/ML algorithm of interest
  - □ Linear/Logistic Regression
  - □ Matrix Factorization
  - □ DNNs
  - □ …



□Express it as a binary circuit

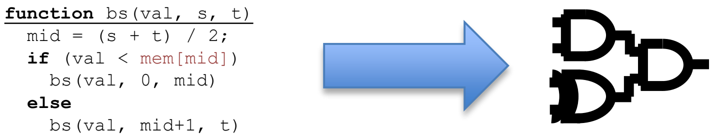


□Apply Yao's Protocol

Caveat: The Three Challenges!

❑Overhead

❑Cost-of-obliviousness



```
function bs(val, s, t)
  mid = (s + t) / 2;
  if (val < mem[mid])
    bs(val, 0, mid)
  else
    bs(val, mid+1, t)
```

❑Maintaining Parallelism

- ❑ Low depth
- ❑ Low fan-out per gate

❑Computational Cost is high!

FPGAs to the rescue!

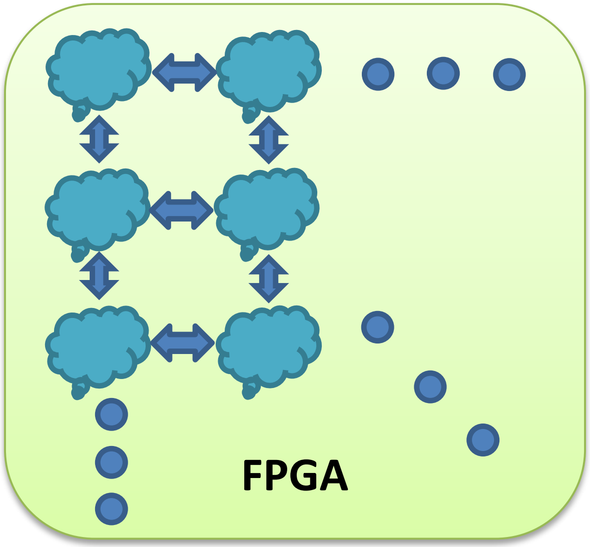
- Initial implementation using a Stratix V
- Currently working on port to AWS

Translating GC to FPGA

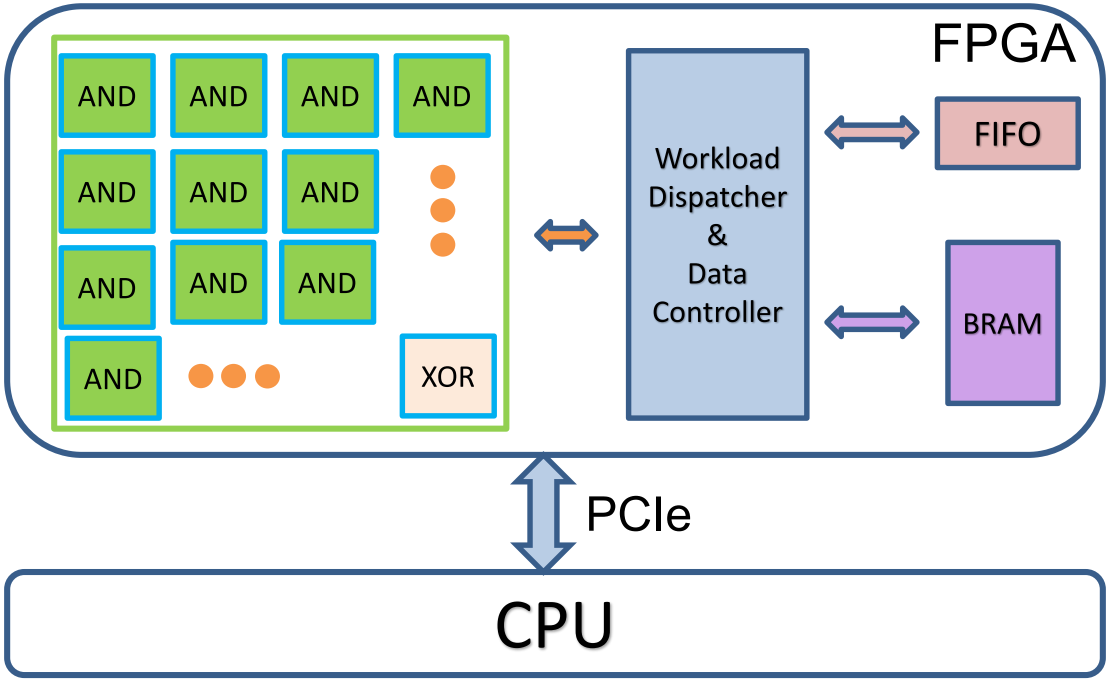
- Translation of problem to Garbled Circuit (GC)
  - Need to do this for SW or hardware
- Translation of GC to FPGA
  - NOT a good approach:
    - Generate GC hardware for each problem instance
      - Can take a ***long*** time
    - Download new circuit when problem changes

FPGA Overlay for Garbled Circuits

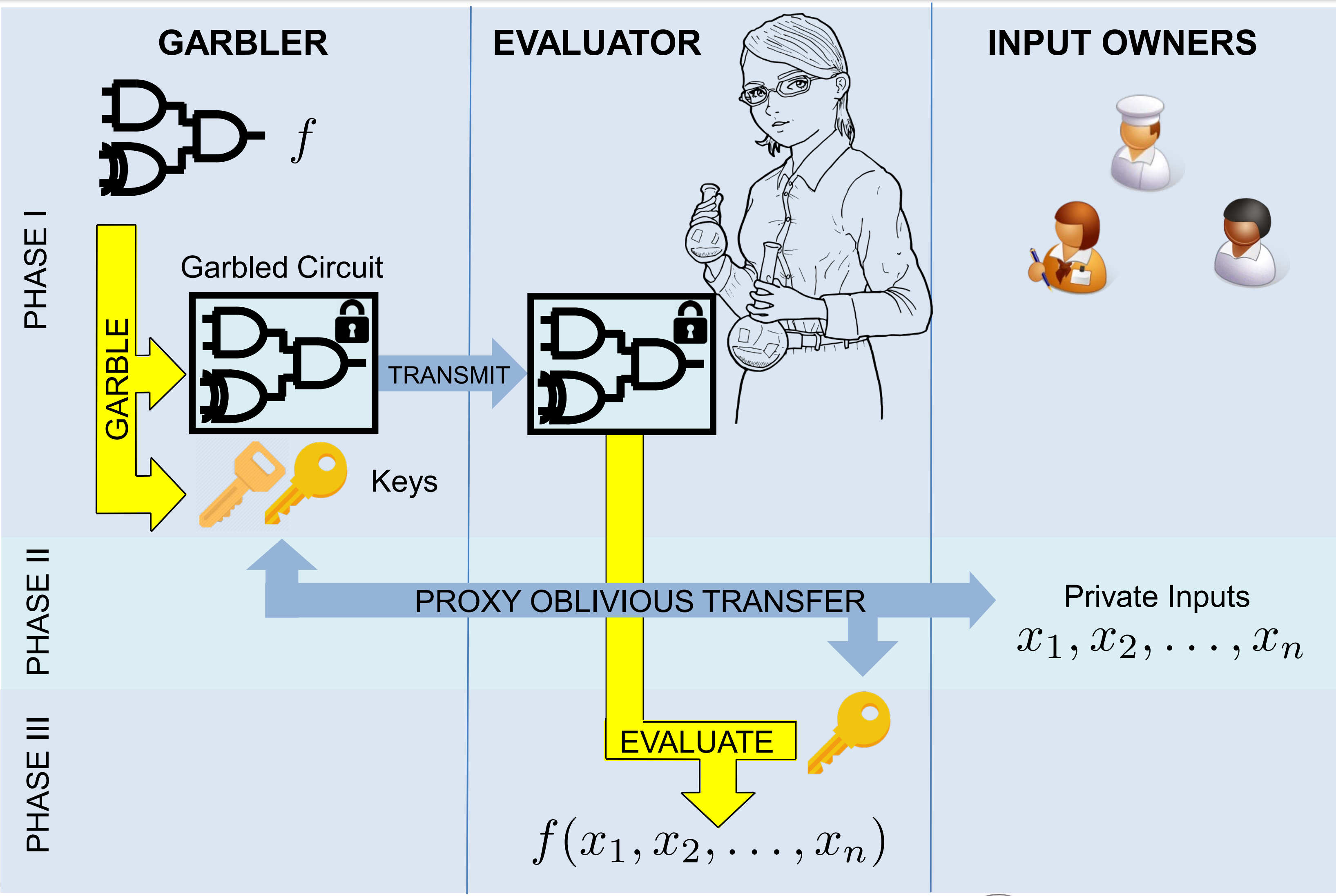
- **Two parts**:
  - A circuit design implemented on FPGA fabric using standard design flow
  - A user circuit mapped onto that overlay circuit
- **Benefits**:
  - User-configured logic in fixed FPGA bitstream
  - Optimized for GC problem (**coarse grain**)
  - One time synthesis and programming
- For GC
  - Garbled AND gates VERY complex
  - Garbled XOR gates somewhat complex



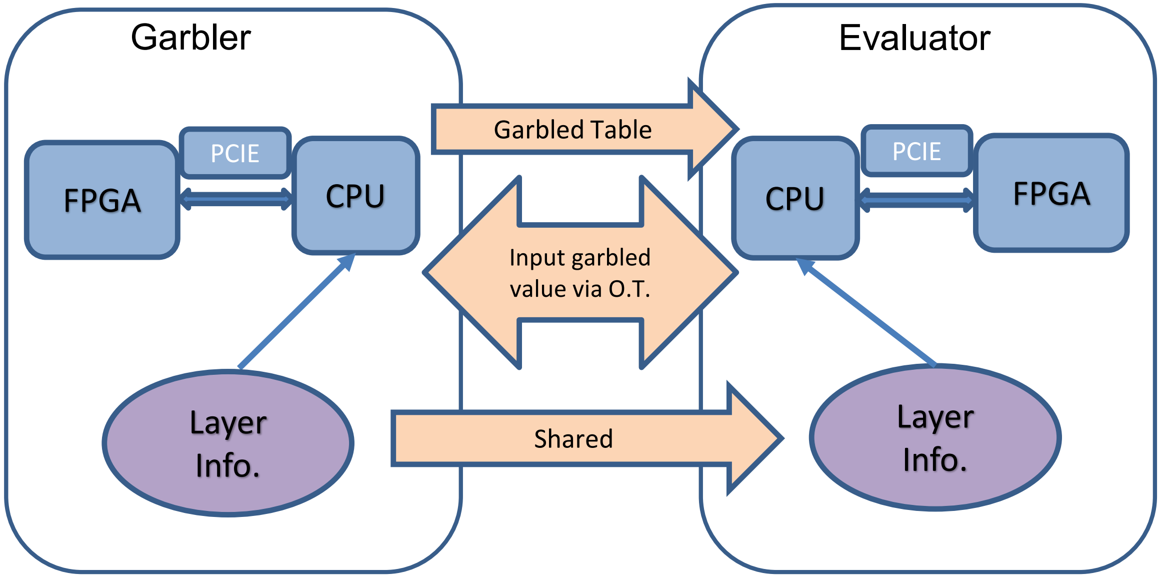
Garbling FPGA



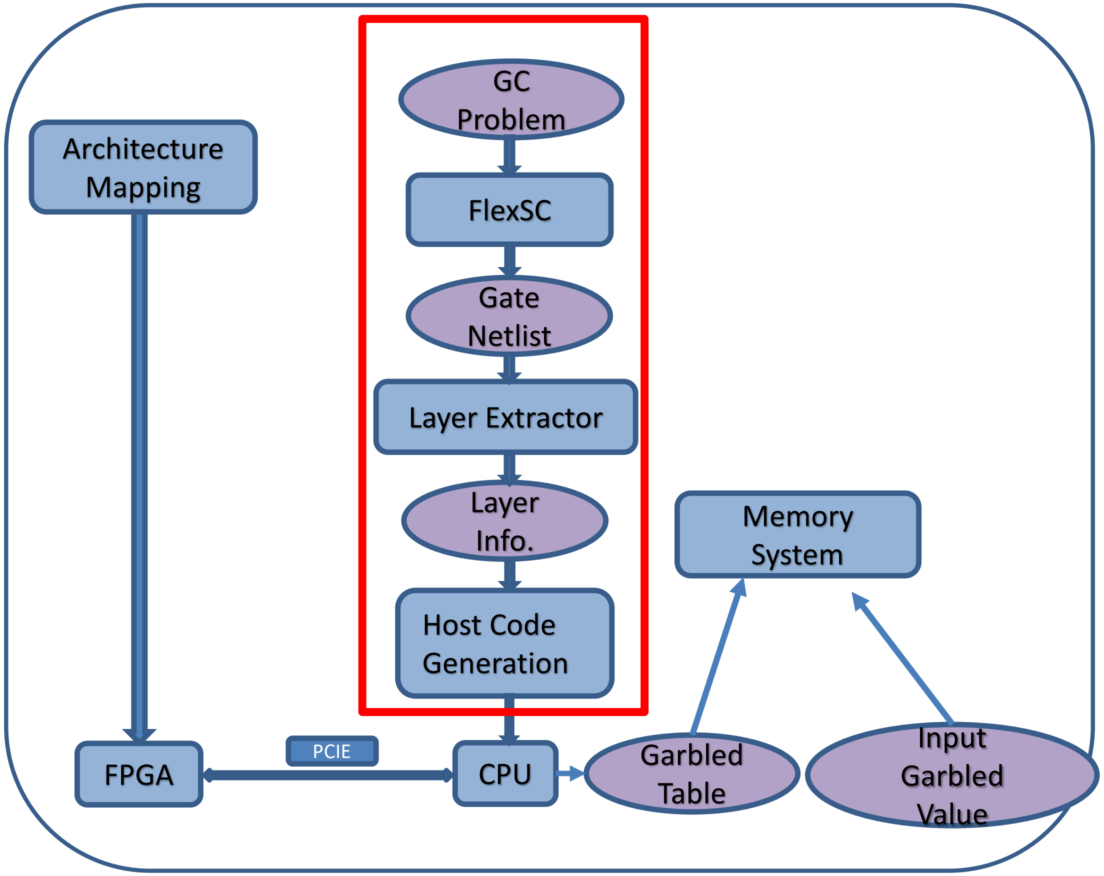
Yao's Garbled Circuits: Protocol Overview



Garbler and Evaluator on FPGAs

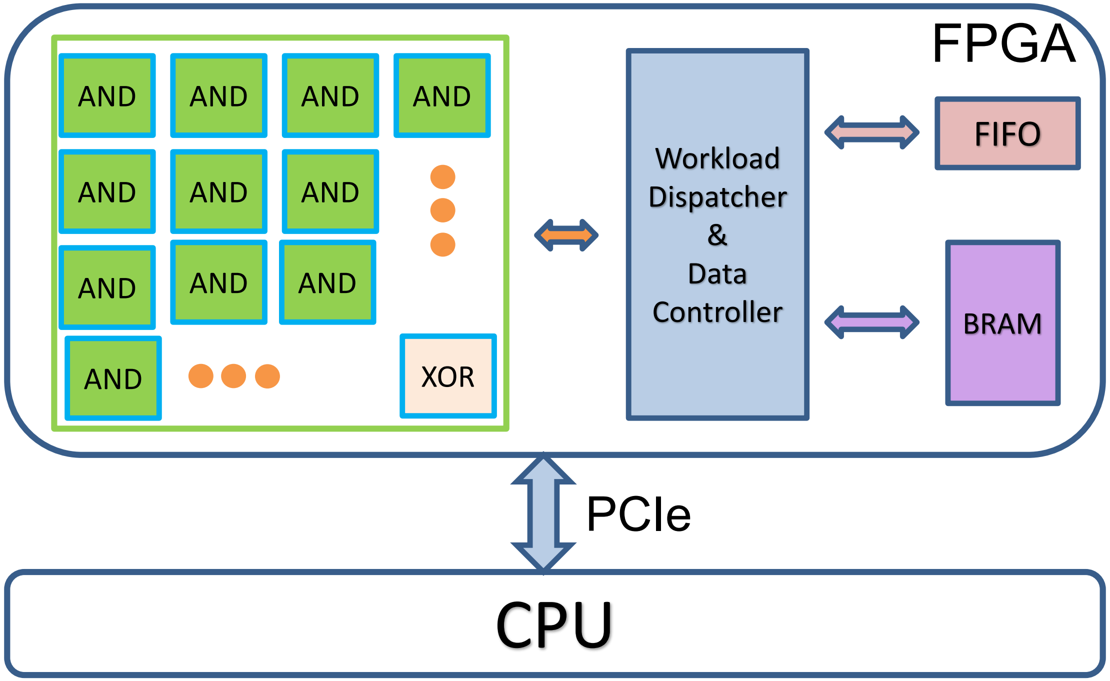


Garbled Circuit Workflow

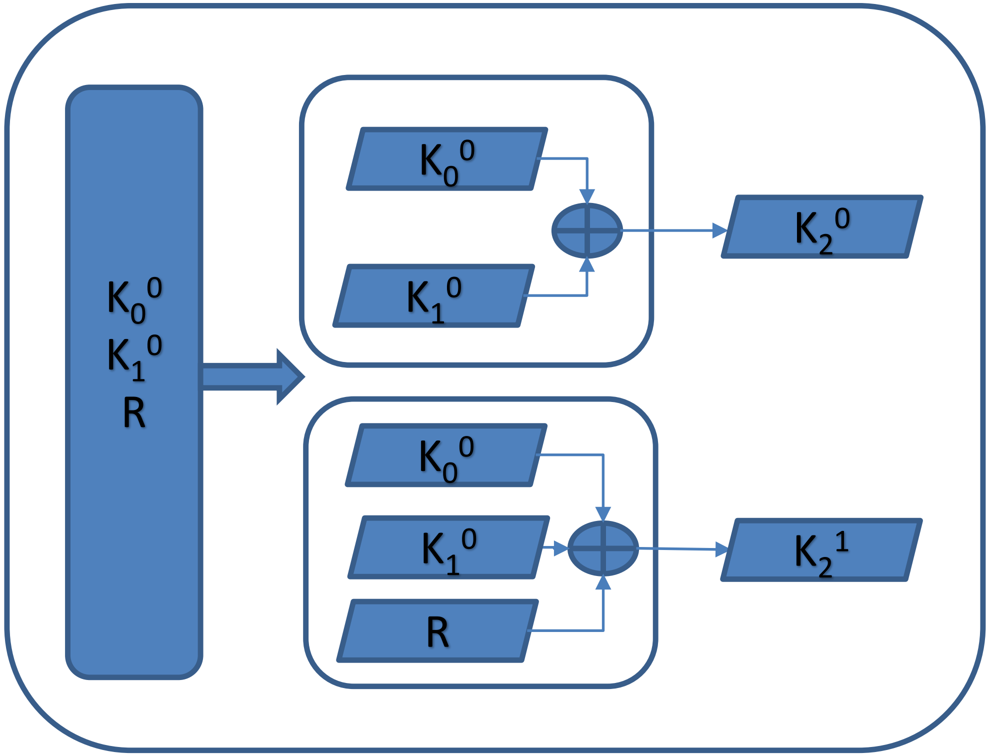


https://github.com/wangxiao1254/FlexSC

Garbling FPGA



Garbled XOR Gate

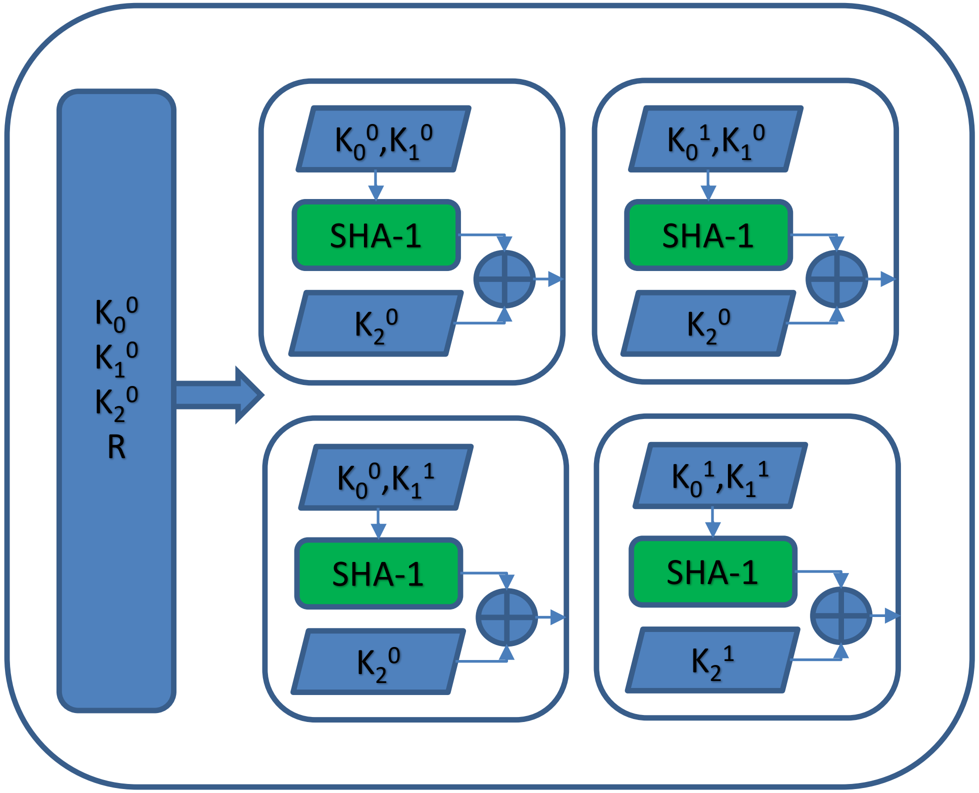


$K_i^{true} = K_i^{false}$ xor R

Latency:
1 cycle

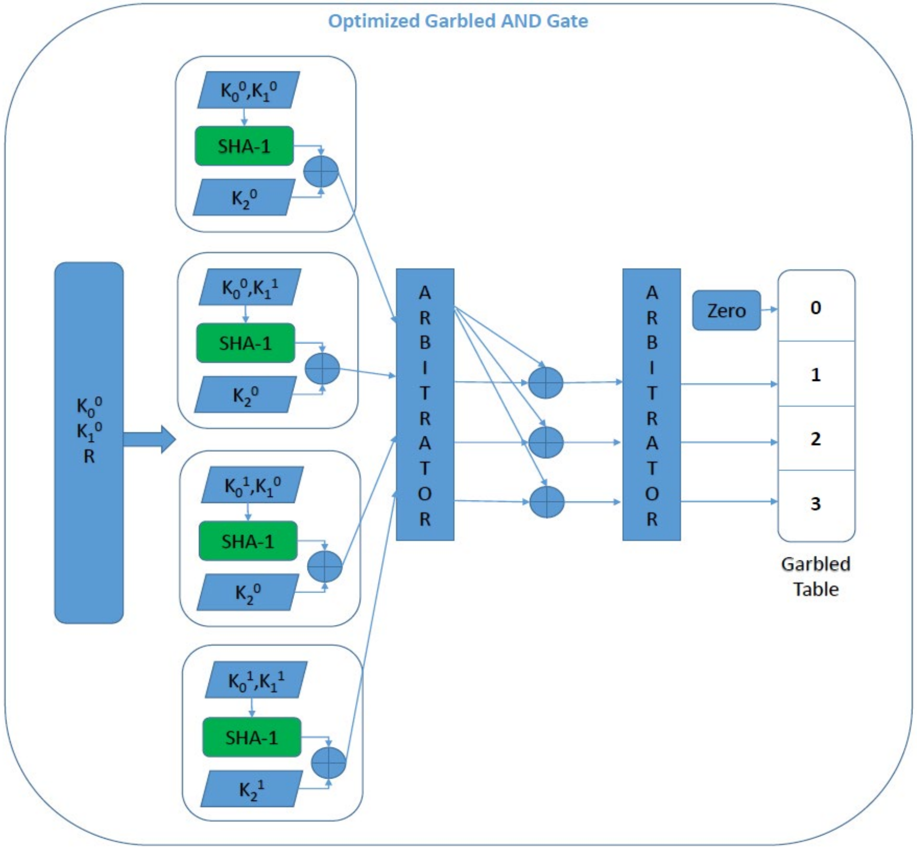
Kolesnikov, Vladimir, and Thomas Schneider. "Improved garbled circuit: Free XOR gates and applications." *International Colloquium on Automata, Languages, and Programming*. 2008.

Basic Garbled AND Gate



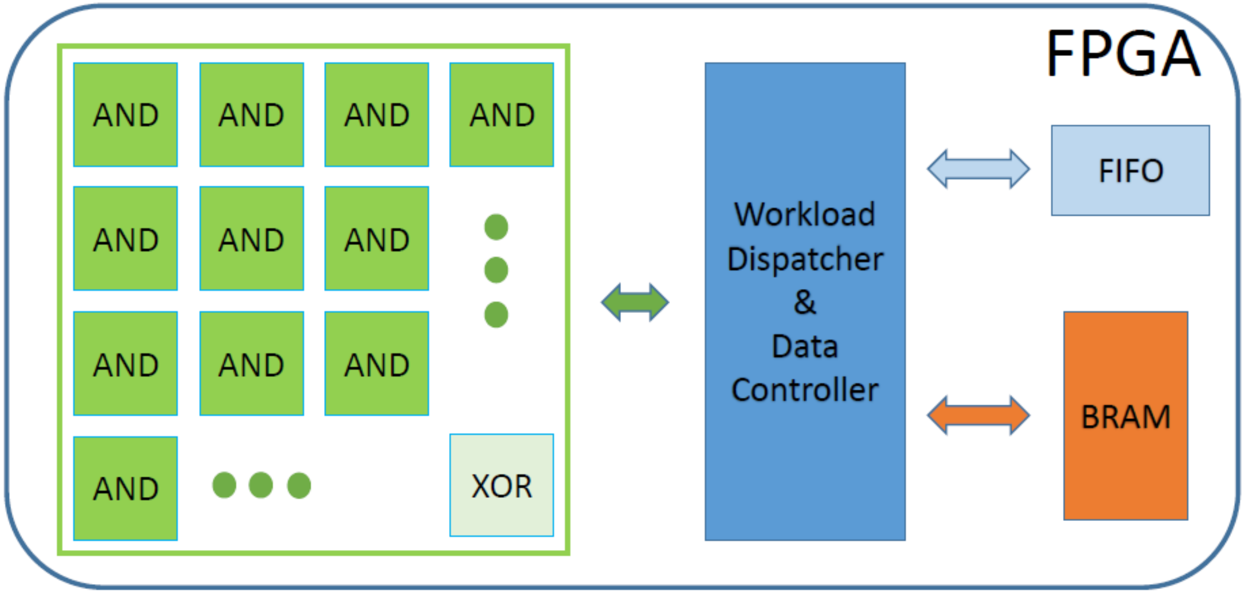
Latency:
82 cycles

Improved Garbled AND Gate



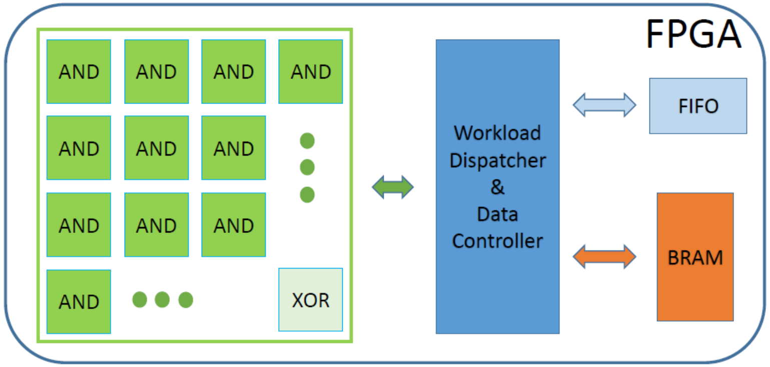
Row Reduction Optimization

Garbler Overlay Architecture Using On Chip Memory

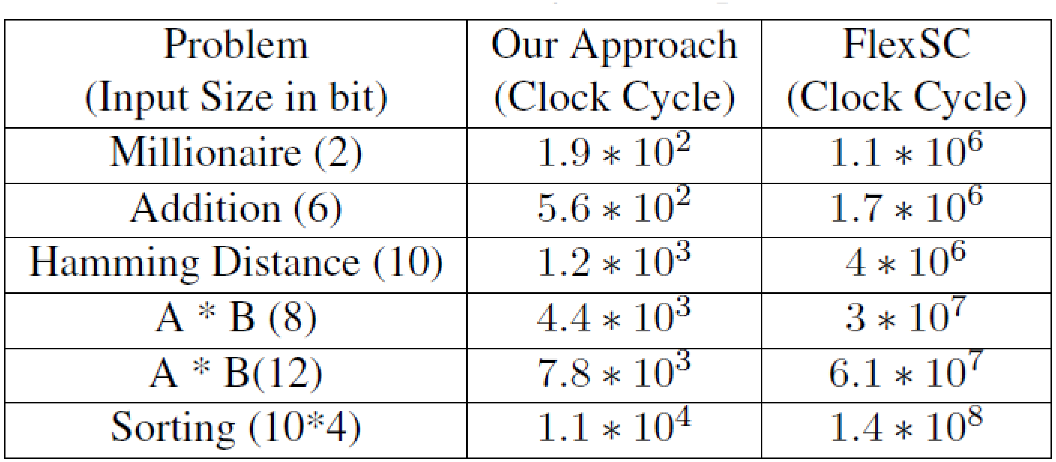


Resource Utilization

Module	ALMs	M20Ks	1-bit Register
One AND	3,070	0	3,750
One XOR	40	0	81
BRAM	0	1,060	0
FIFO	510	280	404
Whole Design	176,893/234,720 75.4%	1,340/2,560 52.3%	215,308

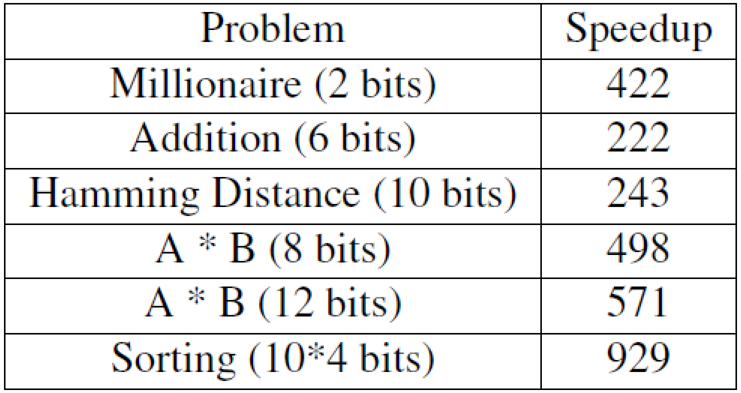


Run times: Our FPGA Implementation vs. FlexSC



Problem (Input Size in bit)	Our Approach (Clock Cycle)	FlexSC (Clock Cycle)
Millionaire (2)	$1.9 * 10^2$	$1.1 * 10^6$
Addition (6)	$5.6 * 10^2$	$1.7 * 10^6$
Hamming Distance (10)	$1.2 * 10^3$	$4 * 10^6$
A * B (8)	$4.4 * 10^3$	$3 * 10^7$
A * B(12)	$7.8 * 10^3$	$6.1 * 10^7$
Sorting (10*4)	$1.1 * 10^4$	$1.4 * 10^8$

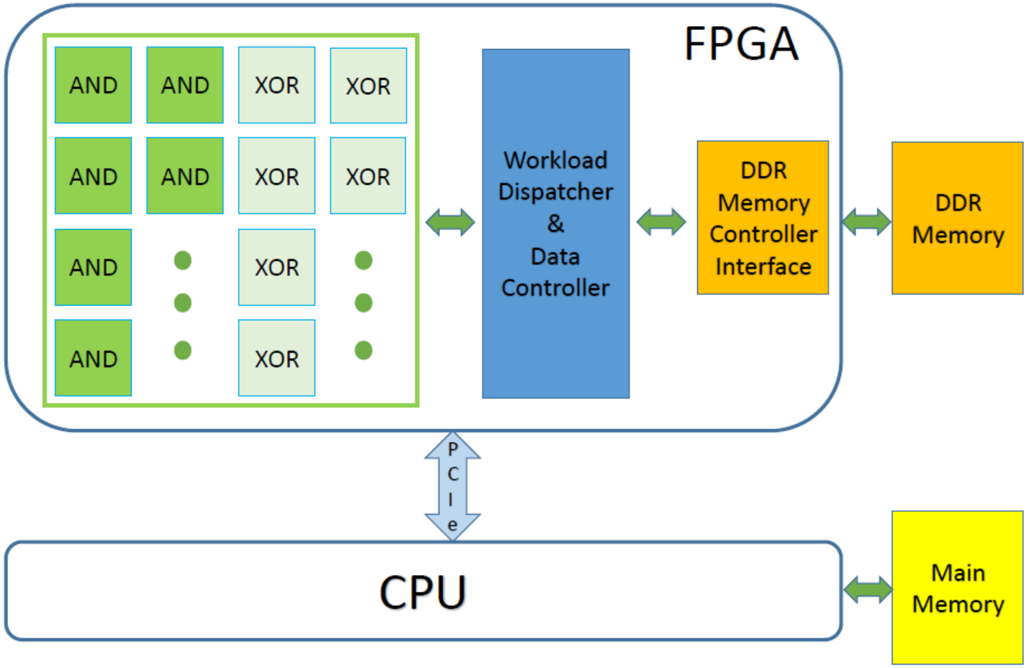
Speedup compared with FlexSC



Problem	Speedup
Millionaire (2 bits)	422
Addition (6 bits)	222
Hamming Distance (10 bits)	243
A * B (8 bits)	498
A * B (12 bits)	571
Sorting (10*4 bits)	929

FlexSC runs at 2.80 GHz
FPGA overlay runs at 200 MHz

Garbler Architecture Using On Board Memory

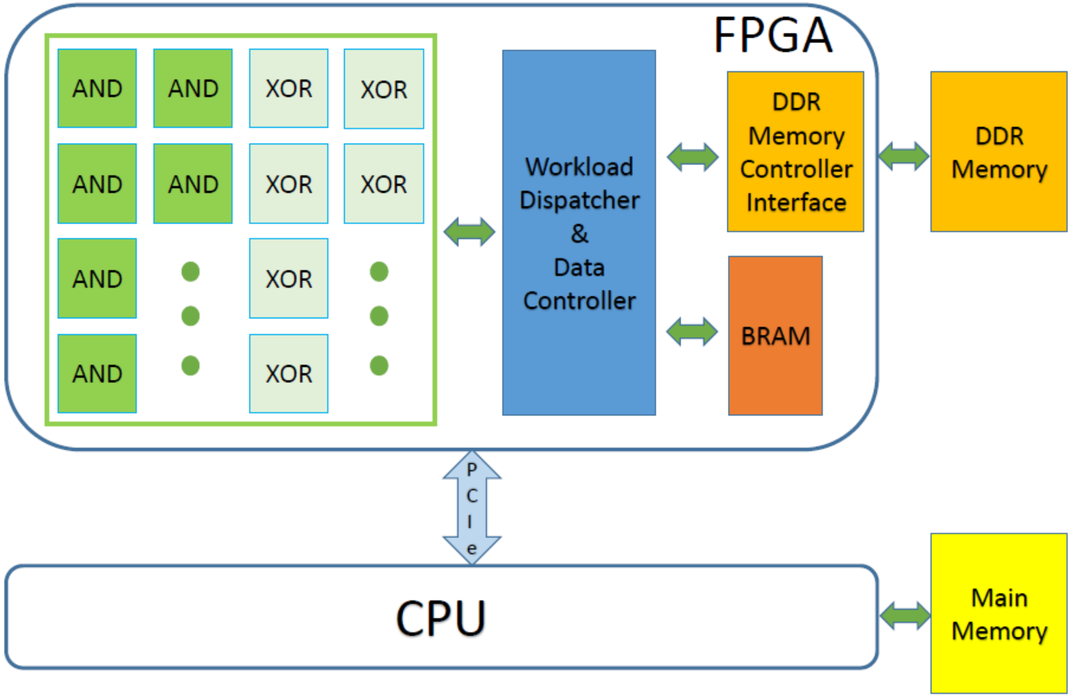

FPGA
AND
AND
XOR
XOR
AND
AND
XOR
XOR
AND
XOR
AND
XOR
Workload Dispatcher & Data Controller
DDR Memory Controller Interface
DDR Memory
PCIe
CPU
Main Memory

For large problems, embedded memory is not large enough to hold all the values

Solution: Use DDR Memory on board

Pro: Memory space
Con: Access speed

Garbler Architecture Using "Caching"



DDR Access is slow compared to embedded memory

Solution: Use BRAM as a user managed "cache"

Pro: Space and Fast

User defined caching policy and implementation

Using on chip memory as user managed cache

- Goal: reduce latency of memory accesses
- Two strategies implemented
  - Most frequently used
    - Count number of uses of each netlist in the entire circuit
    - Store the most frequently used in on-chip memory
  - Directly used
    - Store value in on-chip memory if
      - It is only used once
      - It is used in the next layer

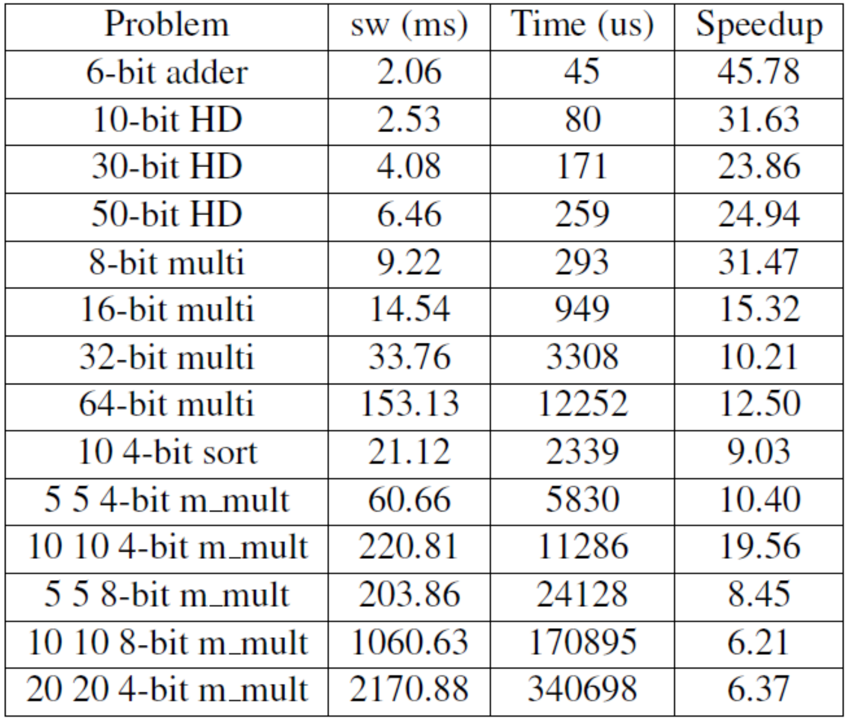
Speedup Results with Multiple Optimizatons

15 AND overlay cells and 15 XOR overlay cells

Hybrid memory system with “directly-used” policy

300 MHz clock for PCIe interface; 200 MHz clock for FPGA fabric

Pipelining and synchronization between host and FPGA optimized



Problem	sw (ms)	Time (us)	Speedup
6-bit adder	2.06	45	45.78
10-bit HD	2.53	80	31.63
30-bit HD	4.08	171	23.86
50-bit HD	6.46	259	24.94
8-bit multi	9.22	293	31.47
16-bit multi	14.54	949	15.32
32-bit multi	33.76	3308	10.21
64-bit multi	153.13	12252	12.50
10 4-bit sort	21.12	2339	9.03
5 5 4-bit m_mult	60.66	5830	10.40
10 10 4-bit m_mult	220.81	11286	19.56
5 5 8-bit m_mult	203.86	24128	8.45
10 10 8-bit m_mult	1060.63	170895	6.21
20 20 4-bit m_mult	2170.88	340698	6.37

Conclusions

- First FPGA overlay architecture to accelerate generic Garbled Circuit problems
- Tools and workflow for modeling and mapping GC problems onto FPGAs
- Speedup compared with state-of-art software platform for arbitrary GC problems

Future Work on FPGA Implementation

- Optimization for better GC performance on one node
  - Expand GC Overlay Cell Library
  - Cross Layer Boolean Operation Extraction

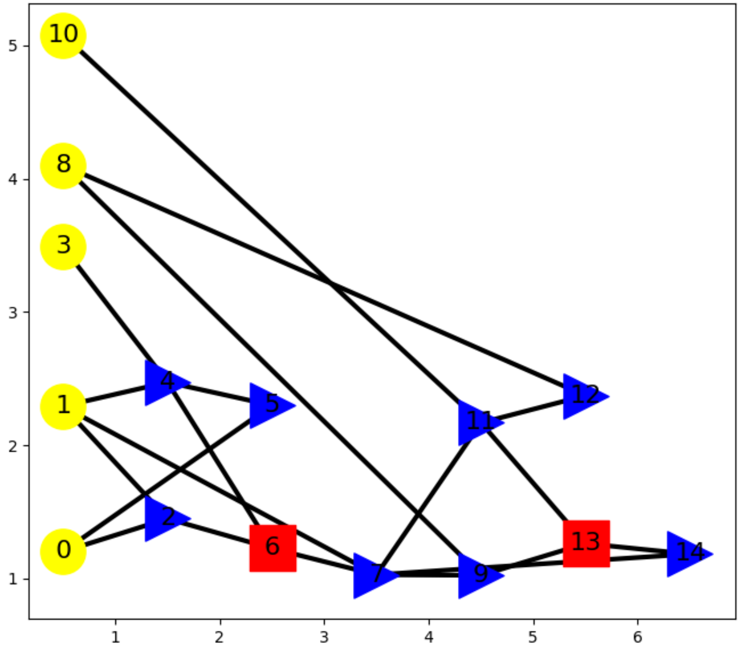
One example:

E = A XOR B
F = C XOR D => G = (A XOR B) AND (C XOR D)
G = E AND F

AWS F1 Design

- CPU communicates with DDR memory
- AXI-4 used to access DDR memory
- AXI-4-lite used to access FPGA registers
- FPGA structure contains garble cores, state machine and logic to access data through AXI-4 and AXI-4-lite
- Challenges
  - Improving latency between host and FPGA
  - Improve latency when accessing off chip memory

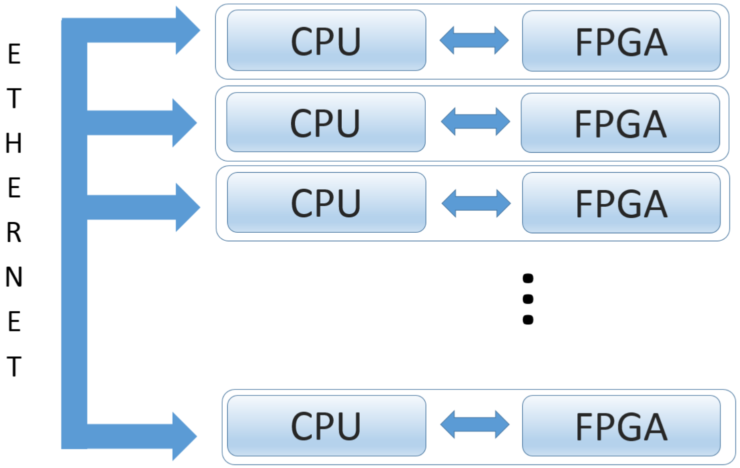
Two bit adder on AWS



- Garbled two bit adder working on AWS
  - Red squares: AND gates
  - Blue triangles: XOR gates
  - Layers shown
- Data transfer time for 256 bytes: 71 us
- Garbled circuit execution time: 41 us
- Design uses off chip memory

Porting to AWS F1 instances, multiple nodes

- Garbled Circuits in the Datacenter with multiple nodes
  - Previous Work done with CPU Cluster[1]
- Challenges:
  - Data Storage
  - Problem Separation
- Port to Amazon Web Services
  - F1 instances: FPGAs in the cloud
- Work supported by NSF funding



[1] GraphSC: Parallel secure computation made easy K Nayak, XS Wang, S Ioannidis, U Weinsberg, N Taft, E Shi 2015 IEEE Symposium on Security and Privacy (SP)

Thank you!

Xin Fang, “Privacy Preserving Computations Accelerated using FPGA Overlays.” PhD dissertation, Northeastern University, August 2017. https://repository.library.northeastern.edu/files/neu:cj82qd893

Xin Fang, Stratis Ioannidis, Miriam Leeser. “Secure Function Evaluation using an FPGA Overlay Architecture”. In International Symposium on Field-Programmable Gate Arrays (FPGA), 2017.

Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. "GraphSC: Parallel secure computation made easy." In IEEE Symposium on Security and Privacy (S&P/OAKLAND), 2015.

Nikolaenko, Valeria, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. "Privacy-preserving matrix factorization." In Conference on Computer & Communications Security (CCS), 2013.