Introduction to Secure Multi-Party Computation

Ryan Moreno

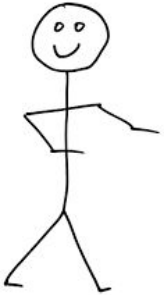
Secure Multi-Party Computation

- Requirements
  - n actors with private data $x_1, x_2, \ldots x_n$
  - compute F($x_1, x_2, \ldots x_n$)
  - don’t leak any other information
  - no trusted third parties
- Applications
  - Distributed voting
  - Private bidding and auctions

The Millionaire Problem - Yao

Do you have more money?

- Don’t leak any other information
- No trusted third-party



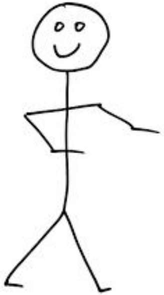
you, a multi-millionaire

me, also a multi-millionaire

The Millionaire Problem - Yao

Does Alice have more money? Effectively: *A* ≥ *B*

- Assume $A, B \in \{1, 2, \ldots 10\}$
- Alice has public RSA key (e, n) and private (d, n)

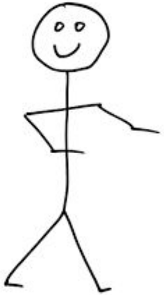


Alice, $*A* Million

Bob, $*B* Million

The Millionaire Problem - Yao

- choose random x such that $|x| = |n|$
- c = encrypt(x) using Alice's public key (e, n)
- $m = c - B + 1 \bmod n$



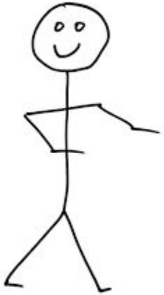
Alice, $*A*$ Million

Bob, $*B*$ Million

The Millionaire Problem - Yao

- choose random x such that $|x| = |n|$
- c = encrypt(x) using Alice's public key (e, n)
- $m = c - B + 1 \bmod n$

← m **looks random**



Alice, $A Million

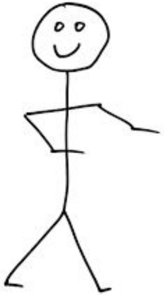
Bob, $B Million

The Millionaire Problem - Yao

- choose random x such that $|x| = |n|$
- c = encrypt(x) using Alice's public key (e, n)
- $m = c - B + 1 \bmod n$

$\leftarrow m$ looks random

- X_i = decrypt($m + i - 1$), $i \in [1, 10]$ $X_B = x$, but all X_i look random



Alice, \$*A* Million

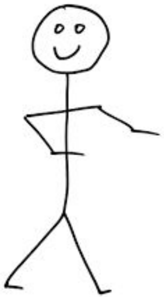
Bob, \$*B* Million

The Millionaire Problem - Yao

- choose random x such that $|x| = |n|$
- c = encrypt(x) using Alice’s public key (e, n)
- $m = c - B + 1 \bmod n$

← m looks random

- X_i = decrypt($m + i - 1$), $i \in [1, 10]$ $X_B = x$, but all X_i look random
- choose a random prime p such that $|p| = |n|/2$ and calculate $X_i \bmod p$ $X_i \bmod p$ all look random
- $W_i = (X_i \bmod p + (i > A)) \bmod p$, $i \in [1, 10]$ add 1 (mod p) iff i is greater than Alice’s wealth



Alice, $A Million

Bob, $B Million

The Millionaire Problem - Yao

- choose random x such that $|x| = |n|$
- c = encrypt(x) using Alice's public key (e, n)
- $m = c - B + 1 \bmod n$

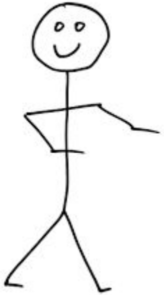
$\leftarrow m$ looks random

- X_i = decrypt($m + i - 1$), $i \in [1, 10]$ $X_B = x$, but all X_i look random
- choose a random prime p such that $|p| = |n|/2$
  and calculate $X_i \bmod p$ $X_i \bmod p$ all look random
- $W_i = (X_i \bmod p + (i > A)) \bmod p$, $i \in [1, 10]$
  add 1 (mod p) iff i is greater than Alice wealth

$p, W_1 \ldots W_{10} \rightarrow$

1 was added to W_B iff $B > A$

W_i looks random and Bob can't tell when 1 was added



Alice, \$*A* Million

Bob, \$*B* Million

The Millionaire Problem - Yao

- choose random x such that $|x| = |n|$
- c = encrypt(x) using Alice's public key (e, n)
- $m = c - B + 1 \bmod n$

$\leftarrow m$ looks random

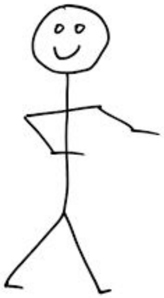
- X_i = decrypt($m + i - 1$), $i \in [1, 10]$ $X_B = x$, but all X_i look random
- choose a random prime p such that $|p| = |n|/2$ and calculate $X_i \bmod p$ $X_i \bmod p$ all look random
- $W_i = (X_i \bmod p + (i > A)) \bmod p$, $i \in [1, 10]$ add 1 (mod p) iff i is greater than Alice's wealth

$p, W_1 \ldots W_{10} \rightarrow$

1 was added to W_B iff $B > A$

W_i looks random and Bob can't tell when 1 was added

- **result = ($W_B \equiv x \pmod{p}$)**



Alice, \$*A* Million

Bob, \$*B* Million

The Millionaire Problem - Yao

- choose random x such that $|x| = |n|$
- c = encrypt(x) using Alice's public key (e, n)
- $m = c - B + 1 \bmod n$

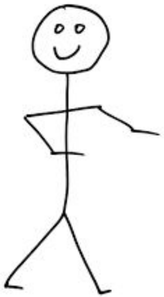
← m looks random

- X_i = decrypt($m + i - 1$), $i \in [1, 10]$ X_B = x, but all X_i look random
- choose a random prime p such that $|p| = |n|/2$ and calculate $X_i \bmod p$ $X_i \bmod p$ all look random
- $W_i = (X_i \bmod p + (i > A)) \bmod p$, $i \in [1, 10]$
  add 1 (mod p) iff i is greater than Alice's wealth

$p, W_1 \ldots W_{10}$ →

1 was added to W_B iff $B > A$

W_i looks random and Bob can't tell when 1 was added



Alice, $\$A$ Million

- result = ($W_B \equiv x \pmod{p}$)
  If $A \geq B$, then 0 added, so
  $W_B = X_B \bmod p = x \bmod p$

← result 1 iff $A \geq B$

Bob, $\$B$ Million

The Millionaire Problem - Yao

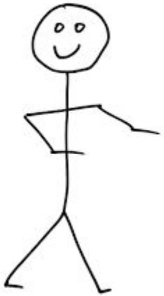
- Correctness
  - result is 1 iff $A \geq B$
- Security
  - Alice learns random number m
  - Bob learns random prime p
  - Bob learns $W_1 \dots W_{10}$
    - Bob can't calculate X_i except when $i = B$, so Bob can't calculate other W_i
    - Bob can't recover X_i from W_i due to loss of information with mod p

The Millionaire Problem - Yao

- Assumptions
    - Actors will follow protocol
    - Actors won’t lie about wealth
    - Actors won’t broadcast their wealth
- Ideal vs. Real World
    - Ideal has a trusted third-party
    - Real world must mimic ideal level of security

Oblivious Transfer (OT)

- Alice offers *n* messages, Bob selects and receives one
  - Alice doesn't know which Bob chose
  - Bob doesn't know the other messages
  - Without loss of generality, we will assume single-bit messages



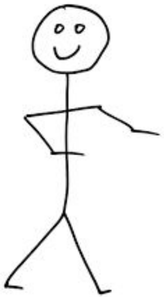
Alice, has $b_1, b_2, \ldots b_n$

Bob, wants b_i

OT - Goldreich, Micali, Widgerson

- choose (f, f^{-1}, B_f) random trapdoor permutation (function, inverse function, hard-core bit)

$f, B_f \rightarrow$



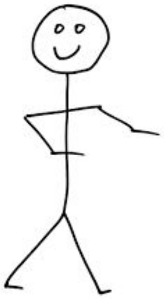
Alice, has $b_1, b_2, \dots b_n$

Bob, wants b_i

OT - Goldreich, Micali, Widgerson

- choose (f, f^{-1}, B_f) random trapdoor permutation (function, inverse function, hard-core bit)

$f, B_f \rightarrow$



- choose random $x_1, x_2, \dots x_n$
- $(y_1, y_2, \dots y_i, \dots y_n) = (x_1, x_2, \dots f(x_i), \dots x_n)$

$\leftarrow (y_1, \dots y_n)$ **looks random**

Alice, has $b_1, b_2, \dots b_n$

Bob, wants b_i

OT - Goldreich, Micali, Widgerson

- choose (f, f^{-1}, B_f) random trapdoor permutation (function, inverse function, hard-core bit)

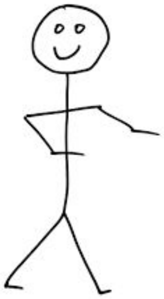
$f, B_f \rightarrow$

- choose random $x_1, x_2, \ldots x_n$
- $(y_1, y_2, \ldots y_i, \ldots y_n) = (x_1, x_2, \ldots f(x_i), \ldots x_n)$

$\leftarrow (y_1, \ldots y_n)$ looks random

- **compute** $(c_1, \ldots c_n) = (B_f(f^{-1}(y_1)), \ldots B_f(f^{-1}(y_n)))$ $c_i = B_f(x_i)$
- **compute** $(d_1, \ldots d_n) = (b_1 \oplus c_1, \ldots b_n \oplus c_n)$ $d_i = b_i \oplus x_i$

looks random $(d_1, \ldots d_n) \rightarrow$



Alice, has $b_1, b_2, \ldots b_n$

Bob, wants b_i

OT - Goldreich, Micali, Widgerson

- choose (f, f^{-1}, B_f) random trapdoor permutation (function, inverse function, hard-core bit)

$f, B_f \rightarrow$

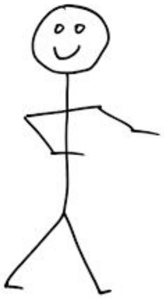
- choose random $x_1, x_2, \dots x_n$
- $(y_1, y_2, \dots y_i, \dots y_n) = (x_1, x_2, \dots f(x_i), \dots x_n)$

$\leftarrow (y_1, \dots y_n)$ looks random

- compute $(c_1, \dots c_n) = (B_f(f^{-1}(y_1)), \dots B_f(f^{-1}(y_n)))$ $c_i = x_i$
- compute $(d_1, \dots d_n) = (b_1 \oplus c_1, \dots b_n \oplus c_n)$ $d_i = b_i \oplus x_i$

looks random $(d_1, \dots d_n) \rightarrow$

- **result = $d_i \oplus x_i$** **result = b_i**



Alice, has $b_1, b_2, \dots b_n$

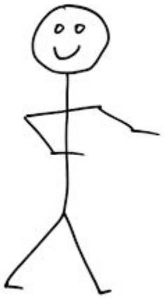
Bob, wants b_i

OT - Goldreich, Micali, Widgerson

- Correctness
  - result is b_i
- Security
  - Alice learns (y_1, ... y_n) which all look random
  - Alice doesn’t learn anything about i
  - Bob learns (d_1, ... d_n) which all look random except d_i
  - Bob can’t calculate any other b_j
    - $d_j = b_j \oplus c_j$
    - c_j calculated with inverse of trapdoor function
    - xor with random loses all information

OT used for simple SMPC

- Alice and Bob have private inputs *x* and *y* respectively
- Want to compute boolean function F(*x*, *y*)

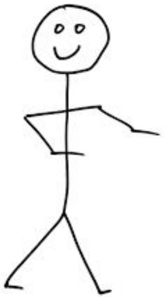


Alice, has *x*

Bob, has *y*

OT used for simple SMPC

- Alice computes b_0 = F(x, 0) and b_1 = F(x, 1)
- Bob uses OT to learn b_y = F(x, y)
- Bob shares the answer with Alice



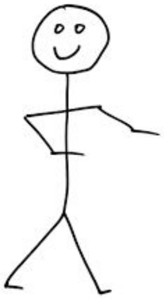
Alice, has x

Bob, has y

OT used for simple SMPC

- Alice computes b_0 = F(x, 0) and b_1 = F(x, 1)
- Bob uses OT to learn b_y = F(x, y)
- Bob shares the answer with Alice

- Consider F(x, y) = $x \land y$
  - Alice has x = 0: F(0, y) doesn't leak y
  - Bob has y = 0: F(x, 0) doesn't leak x
  - Alice has x = 1: F(1, y) leaks y
  - Holds up to security of ideal world



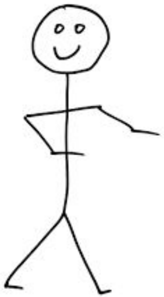
Alice, has x

Bob, has y

OT used for simple SMPC

- Alice computes b_0 = F(x, 0) and b_1 = F(x, 1)
- Bob uses OT to learn b_y = F(x, y)
- Bob shares the answer with Alice

- Single-gate, single-bit boolean functions only
  - Otherwise Alice would gain information at each individual OT

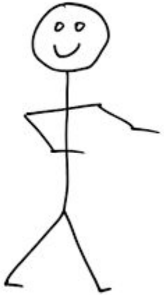


Alice, has x

Bob, has y

OT used for SMPC

- Alice and Bob have private inputs *x* and *y* respectively
- Want to compute boolean function F(*x*, *y*) where F consists of multiple gates and *x* and *y* are multiple bits
    - Each step will consider a single gate with single-bit inputs f(*a*, *b*) with the output encoded



Alice, has *x*

Bob, has *y*

OT used for SMPC

- create encryption schemes $S_1 = (E_1, D_1)$ to S_6
- randomly select *p*, *s*, *m*, and *u*
- randomly assign S_3 and S_4 complimentary bits
- randomly assign S_5 and S_6 complimentary bits
- create table for f(*a*, *b*)

a = 0: S_1
a = 1: S_2

b = 0: S_3
b = 1: S_4

← table with rows permuted and no private values
← D_3 or D_4 dependent on *b*

S_1	$E_1(p)$	S_3	$E_3(q)$
S_1	$E_1(s)$	S_4	$E_4(t)$
S_2	$E_2(m)$	S_3	$E_3(n)$
S_2	$E_2(u)$	S_4	$E_4(v)$

0: S_5
1: S_6

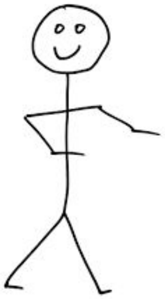
Example: F(*a*, *b*) = $a \wedge b$

$p \oplus q = D_5$ $(0 \wedge 0 = 0)$

$s \oplus t = D_5$ $(0 \wedge 1 = 0)$

$m \oplus n = D_5$ $(1 \wedge 0 = 0)$

$u \oplus v = D_6$ $(1 \wedge 1 = 1)$



Alice, has *a*

Bob, has *b*

OT used for SMPC

- create encryption schemes $S_1 = (E_1, D_1)$ to S_6
- randomly select *p*, *s*, *m*, and *u*
- randomly assign S_3 and S_4 complimentary bits
- randomly assign S_5 and S_6 complimentary bits
- create table for f(*a*, *b*)

a = 0: S_1
a = 1: S_2

b = 0: S_3
b = 1: S_4

S_1	$E_1(p)$	S_3	$E_3(q)$
S_1	$E_1(s)$	S_4	$E_4(t)$
S_2	$E_2(m)$	S_3	$E_3(n)$
S_2	$E_2(u)$	S_4	$E_4(v)$

0: S_5
1: S_6

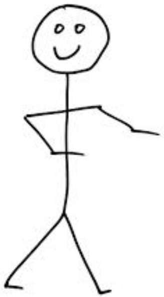
← table with rows permuted and no private values

← D_3 or D_4 dependent on *b*

← **D_1 or D_2 sent using OT dependent on *a***

Example: F(*a*, *b*) = *a* ∧ *b*

$p \oplus q = D_5$ (0 ∧ 0 = 0)
$s \oplus t = D_5$ (0 ∧ 1 = 0)
$m \oplus n = D_5$ (1 ∧ 0 = 0)
$u \oplus v = D_6$ (1 ∧ 1 = 1)



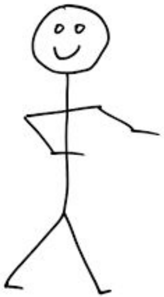
Alice, has *a*

Bob, has *b*

OT used for SMPC

- use the pair of decryption keys to decode the pair of values *k*, *l* in a row
- $D_i = k \oplus l$ $D_i = D_5$ or D_6
- result = 0 if D_5, 1 otherwise result = f(*a*, *b*)

result →



Alice, has *a*

- create encryption schemes and table

← table with rows permuted and no private values

← D_3 or D_4 dependent on *b*

← D_1 or D_2 sent using OT dependent on *a*

a = 0: S_1

a = 1: S_2

b = 0: S_3

b = 1: S_4

S_1	$E_1(p)$	S_3	$E_3(q)$
S_1	$E_1(s)$	S_4	$E_4(t)$
S_2	$E_2(m)$	S_3	$E_3(n)$
S_2	$E_2(u)$	S_4	$E_4(v)$

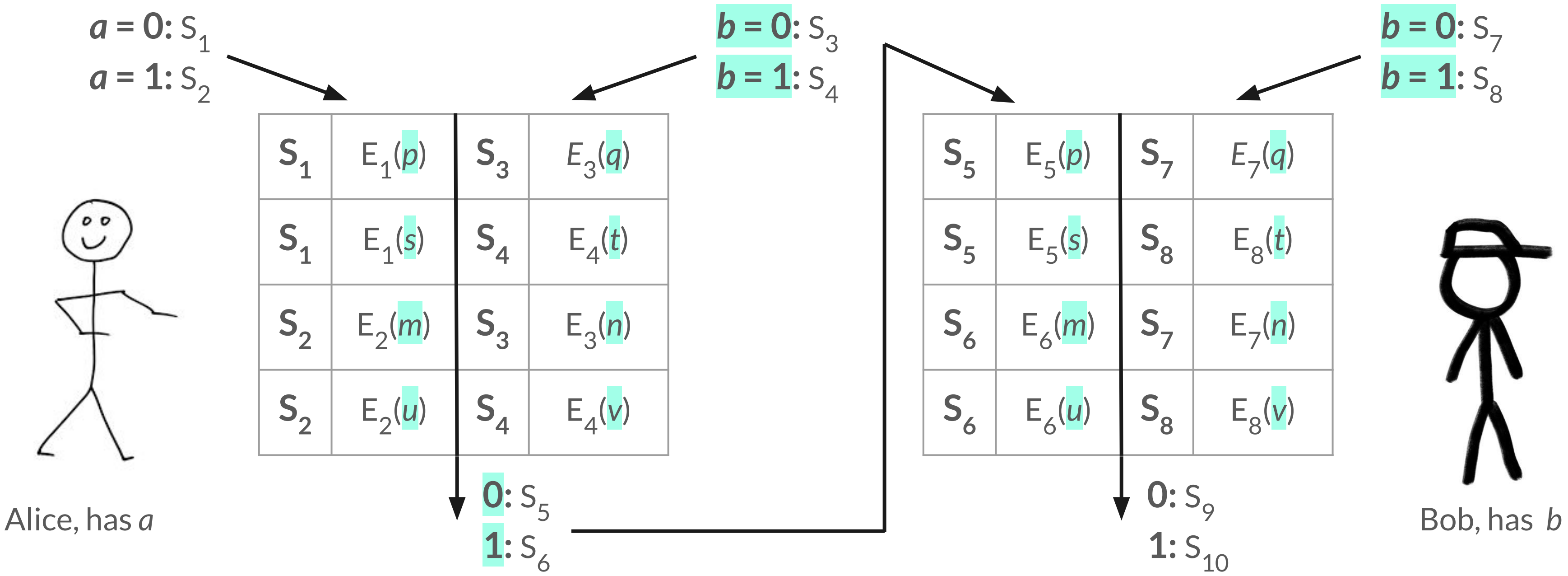
0: S_5

1: S_6

Bob, has *b*

OT used for SMPC

- combine single-bit, single-gate steps
    - keep intermediate output assignments private
    - Use intermediate outputs as inputs



a = 0: S_1
a = 1: S_2

***b* = 0**: S_3
***b* = 1**: S_4

S_1	$E_1(p)$	S_3	$E_3(q)$
S_1	$E_1(s)$	S_4	$E_4(t)$
S_2	$E_2(m)$	S_3	$E_3(n)$
S_2	$E_2(u)$	S_4	$E_4(v)$

0: S_5
1: S_6

***b* = 0**: S_7
***b* = 1**: S_8

S_5	$E_5(p)$	S_7	$E_7(q)$
S_5	$E_5(s)$	S_8	$E_8(t)$
S_6	$E_6(m)$	S_7	$E_7(n)$
S_6	$E_6(u)$	S_8	$E_8(v)$

0: S_9
1: S_{10}

OT used for SMPC

- Correctness
  - result of each step is f(*x*, *y*)
    - final result is F(*a*, *b*)
  - any boolean function can be composed with $\wedge$ and $\neg$
- Security
  - Alice learns either D_3 or D_4, uncorrelated with *b*
  - Alice learns only D_1 or D_2, according to *a*
  - Alice can only compute either D_5 or D_6 with both *k* and *l*
    - xor with random renders partial information useless
  - Alice doesn't learn intermediate outputs because correlation is private
  - Bob learns only the final result
  - Bob doesn't learn intermediate outputs because no information transfer

Secure Multi-Party Computation

- Recap
  - we’ve shown any boolean function can be securely computed
  - constraints - two actors, passive adversaries
- Goldreich, Micali, and Widgerson proved completeness for *n* actors
  - can have malicious adversaries provided at least *n*/2 are honest

Works Cited

A. C. Yao, “Protocols for Secure Computations,” in *SFCS ’82 Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, 1982, pp. 160-164.

O. Goldreich, S. Micali, and A. Wigderson, “How to Play ANY Mental Game,” in *STOC ’87 Proceedings of the nineteenth annual ACM Symposium on Theory of Computing*, 1987, pp. 218-229.

M. J. Fischer, “Lecture Notes: CPSC 461b: Foundations of Cryptography.” *Yale University Department of Computer Science*, 2009.