Secure Multi-Party Computation Tutorial

Fattaneh Bayatbabolghani* **and Marina Blanton**†

*University of California, Berkeley

†University at Buffalo, The State University of New York

ACM Conference on Computer and Communications Security

October 2018

Outline

- Introduction
  - motivation
  - security definition
- Garbled circuit evaluation
  - Yao's protocol
  - oblivious transfer and its extensions
  - garbled circuit optimizations
  - malicious adversary techniques

Outline

- Secret sharing
  - Shamir secret sharing
  - operations on shares
  - malicious adversary techniques
- Compilers
  - secure two-party compilers
  - secure multi-party compilers
- Summary

Data Privacy

- Why do we talk about protecting data privacy?

Data Privacy

- Larger and larger volumes of data are being collected about individuals
  - one's shopping behavior, geo location and moving patterns, interests and hobbies, exercise patterns, etc.
- Even intended analysis and use of data is scary, but it is also prone to abuse
  - information about individuals collected by an entity can be legitimately sold to others
  - large datasets with sensitive information are an attractive target for insider abuse
  - data breaches are more common than what we know

Data Protection

- There are many different ways to protect private, proprietary, classified or otherwise sensitive information
  - this tutorial will cover some of such techniques
- Protection techniques include:
  - computing on private data without revealing the data
  - anonymous communication and authentication
  - applications that provide anonymity (e-cash, voting, etc.)

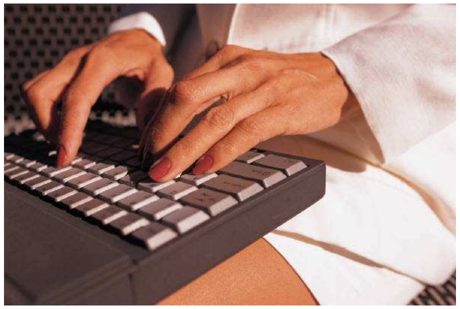
Secure Multi-Party Computation

- Secure multi-party computation allows two or more individuals to jointly evaluate a function on their respective private data
  - security guarantees allow for no unintended information leakage
  - only output of the computation (and any information deduced from the output and individual private input) can be known to a participant

Example Secure Two-Party Computation

- Two millionaires Alice and Bob would like to determine who is richer without revealing their worth to each other

Alice
private x



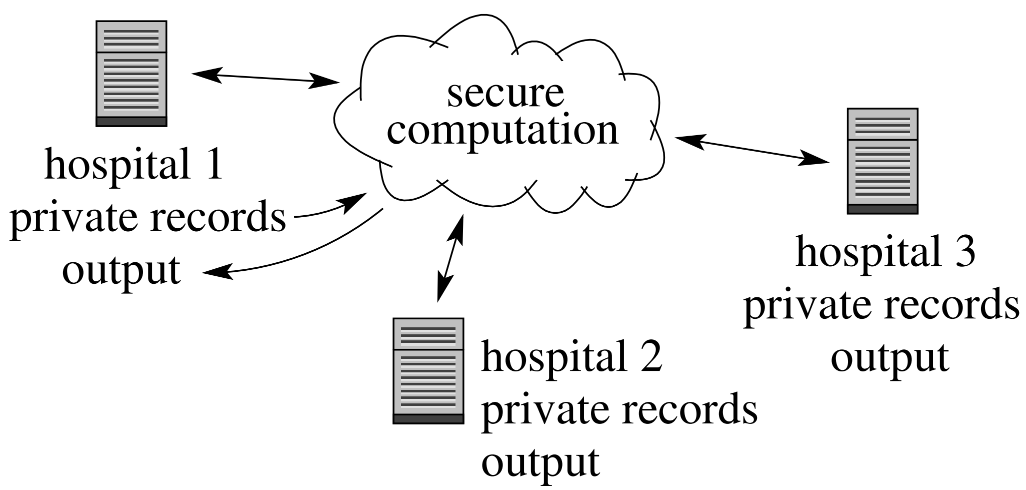
$\longrightarrow$
$\longleftarrow$
$\longrightarrow$
$\longleftarrow$
$\longrightarrow$

Bob
private y

output
$x < y$

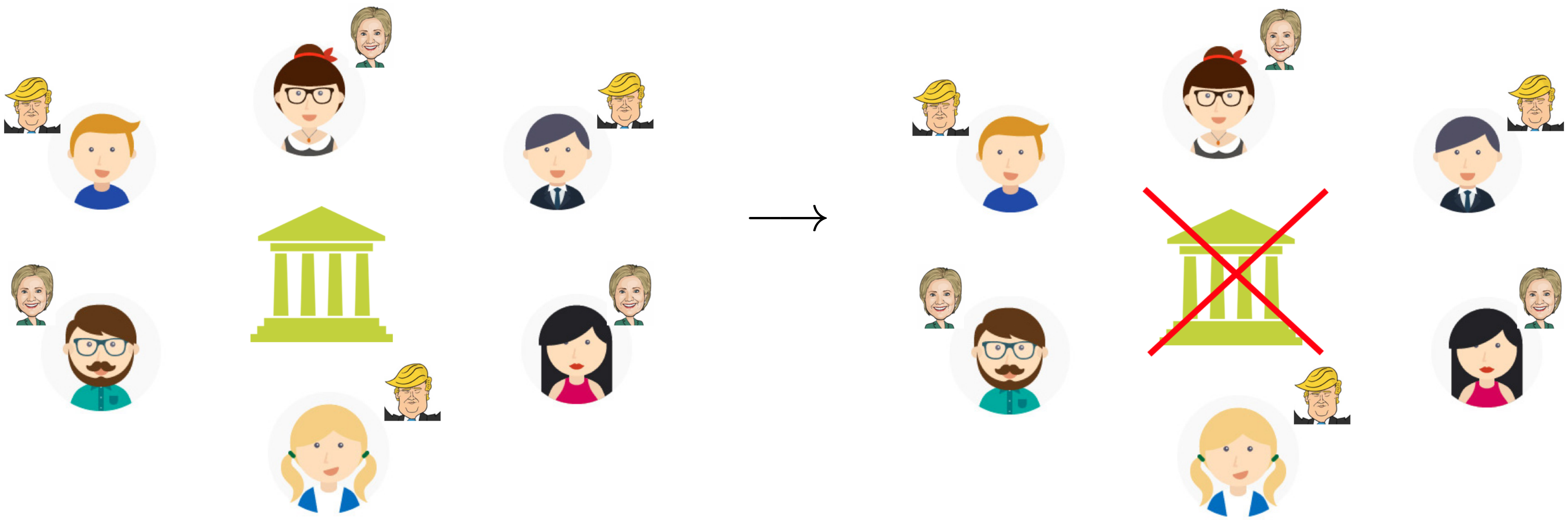
Example Secure Multi-Party Computation

- A number of local hospitals would like to jointly determine the most effective treatment to a rare disease



Example Secure Multi-Party Computation

- Many individuals would participate in electronic voting



- Any computation that can be done with a trusted third party (TTP) can be done without TTP

Secure Multi-Party Computation

- Regardless of the setup, the same strong security guarantees are expected:
  - suppose there is an ideal third party that the participants trust with their data
  - they send their data to TTP and receive the output
  - then a multi-party protocol is secure if adversarial participants learn no more information than in the case of ideal TTP
  - this is formalized through a simulation paradigm

Security of SMC

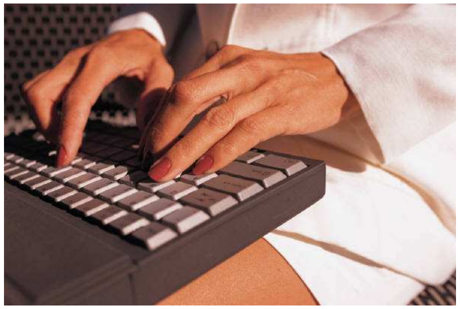
- There are two standard ways of modeling participants in SMC
  - a semi-honest participant complies with the prescribed computation, but might attempt to learn additional information about other participants' data from the messages it receives
    - it is also called honest-but-curious or passive
  - a malicious participant can arbitrarily deviate from the protocol's execution in the attempt to learn unauthorized information about other participants' data
    - it is also called active
- There is a third type of adversarial model with covert participants who can act maliciously, but do not wish to be caught

Security of SMC in the Semi-Honest Model

- We start modeling security using the semi-honest model
  - Let n be the number of participants in secure computation
  - An adversary $\mathcal{A}$ can corrupt and control $t < n$ of them
  - $\mathcal{A}$ knows all information that the corrupt parties have and receive
  - Security is modeled by building a simulator $S_\mathcal{A}$ with access to the TTP that produces $\mathcal{A}$'s view indistinguishable from its view in real protocol execution
    - $S_\mathcal{A}$ has $\mathcal{A}$'s information and TTP's output
    - it must simulate the view of $\mathcal{A}$ and form outputs for all parties correctly

The Real Model

Alice
private x



$\longrightarrow$
$\longleftarrow$
$\longrightarrow$
$\longleftarrow$
$\longrightarrow$

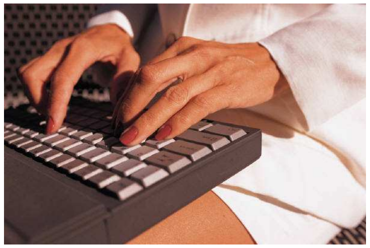
Bob
private y

protocol output

protocol output

The Ideal Model

Alice
private x



TTP

$\longrightarrow$ $\longleftarrow$ $\longrightarrow$ $\longleftarrow$ $\longrightarrow$

$\longrightarrow$ $\longleftarrow$ $\longrightarrow$ $\longleftarrow$ $\longrightarrow$

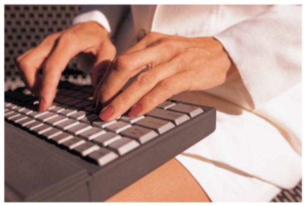
Bob
private y

protocol output

protocol output

The Security Definition

Adversary S



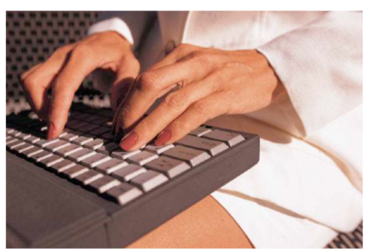
$\longrightarrow$
$\longleftarrow$
$\longrightarrow$
$\longleftarrow$
$\longrightarrow$

$\longrightarrow$
$\longleftarrow$
$\longrightarrow$
$\longleftarrow$
$\longrightarrow$

The Ideal Model

$\simeq$

Adversary A



$\longrightarrow$
$\longleftarrow$
$\longrightarrow$
$\longleftarrow$
$\longrightarrow$

The Real Model

Security of SMC in the Semi-Honest Model

- Formal definition:
  - Let parties $P_1, \ldots, P_n$ engage in a protocol Π that computes function $f(\mathsf{in}_1, \ldots, \mathsf{in}_n) \rightarrow (\mathsf{out}_1, \ldots, \mathsf{out}_n)$, where $\mathsf{in}_i \in \{0, 1\}^*$ and $\mathsf{out}_i \in \{0, 1\}^*$ denote the input and output of party P_i, respectively.
  - Let $\mathrm{VIEW}_\Pi(P_i)$ denote the view of participant P_i during the execution of protocol Π. That is, P_i's view is formed by its input and internal random coin tosses r_i, as well as messages $m_1, \ldots, m_k$ passed between the parties during protocol execution:

$$\mathrm{VIEW}_\Pi(P_i) = (\mathsf{in}_i, r_i, m_1, \ldots, m_k).$$

  - Let $I = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\}$ denote a subset of the participants for $t < n$ and $\mathsf{VIEW}_\Pi(I)$ denote the combined view of participants in I during the execution of protocol Π (i.e., the union of the views of the participants in I).

Security of SMC in the Semi-Honest Model

- Formal definition (cont.):
  - We say that protocol Π is t-private in the presence of semi-honest adversaries if for each coalition of size at most t there exists a probabilistic polynomial time simulator S_I such that

$$S_I(\mathsf{in}_I, f(\mathsf{in}_1, \ldots, \mathsf{in}_n)) \equiv \{\text{VIEW}_{\Pi}(I), \mathsf{out}_I\},$$

    where $\mathsf{in}_I = \bigcup_{P_i \in I}\{\mathsf{in}_i\}$, $\mathsf{out}_I = \bigcup_{P_i \in I}\{\mathsf{out}_i\}$, and $\equiv$ denotes computational or statistical indistinguishability.

- Computational indistinguishability of two distributions means that the probability that they differ is negligible in the security parameter κ
  - for statistical indistinguishability, the difference must be negligible in the statistical security parameter

Security of SMC in the Malicious Model

- In the malicious model we have the following definition:
  - Let Π be a protocol that computes function $f(\mathsf{in}_1, \ldots, \mathsf{in}_n) \rightarrow (\mathsf{out}_1, \ldots, \mathsf{out}_n)$, with party P_i contributing input $\mathsf{in}_i \in \{0, 1\}^*$ and receiving output $\mathsf{out}_i \in \{0, 1\}^*$
  - Let $\mathcal{A}$ be an arbitrary algorithm with auxiliary input x and S be an adversary/simulator in the ideal model
  - Let $\text{REAL}_{\Pi,\mathcal{A}(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ denote the view of adversary $\mathcal{A}$ controlling parties in I together with the honest parties' outputs after real protocol Π execution
  - Similarly, let $\text{IDEAL}_{f,S(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ denote the view of S and outputs of honest parties after ideal execution of function f

Security of SMC in the Malicious Model

- Formal definition (cont.):
  - We say that Π t-securely computes f if for each coalition I of size at most t, every probabilistic adversary $\mathcal{A}$ in the real model, all $\mathsf{in}_i \in \{0,1\}^*$ and $x \in \{0,1\}^*$, there is probabilistic S in the ideal model that runs in time polynomial in $\mathcal{A}$'s runtime and

$$\{\text{IDEAL}_{f,S(x),I}(\mathsf{in}_1,\ldots,\mathsf{in}_n)\} \equiv \{\text{REAL}_{\Pi,\mathcal{A}(x),I}(\mathsf{in}_1,\ldots,\mathsf{in}_n)\}$$

Secure Multi-Party Computation

- The setting can be further generalized to allow for more general setups
- We can distinguish between three groups of participants
  - input parties (data owners) contribute their private input into the computation
  - computational parties securely execute the computation on behalf of all participants
  - output parties (output recipients) receive output from the computational parties at the end of the computation
- The groups can be arbitrarily overlapping

Secure Multi-Party Computation

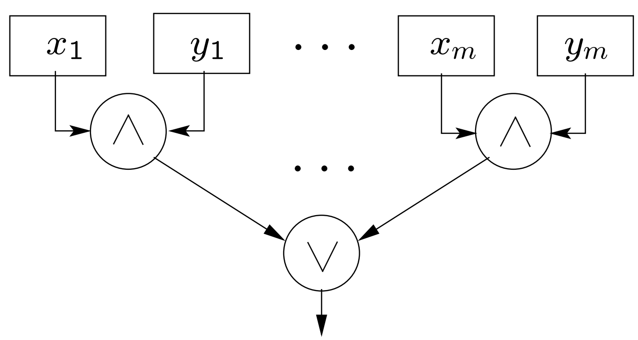
- The above setup allows for many interesting settings
  - e.g., a large number of participating hospitals can choose a subset of them to run the computation on behalf of all of them
  - they can also employ external parties (cloud providers) for running the computation
  - the output can be delivered to a subset of them and/or to other interested parties
- This setup also allows for secure computation outsourcing
  - one or more clients securely outsource their computation to a number of external cloud computing providers

Secure Multi-Party Computation Techniques

- Garbled circuit evaluation
  - two-party computation ($n = 2$)
- Linear secret sharing
  - multi-party computation ($n > 2$)
- Homomorphic encryption
  - two- or multi-party computation ($n \geq 2$)

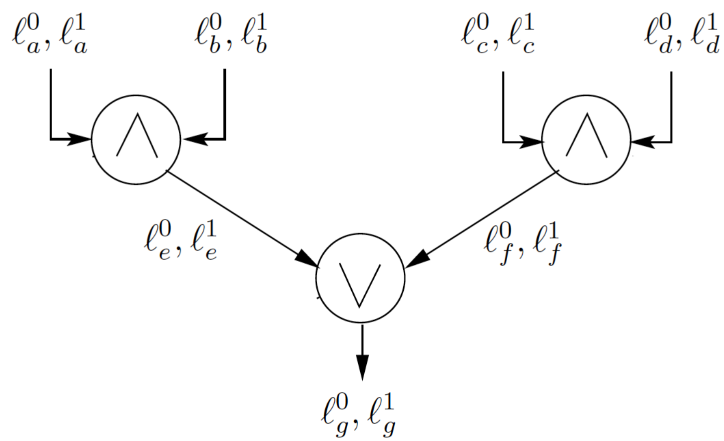
Garbled Circuit Evaluation

- SMC based on garbled circuit evaluation involves two participants: circuit garbler and circuit evaluator
- The function to be computed is represented as a Boolean circuit
  - typically we'll use binary (two input and one output bits) gates and negation gates
  - example:



Yao's Protocol: Garbled Circuit Evaluation

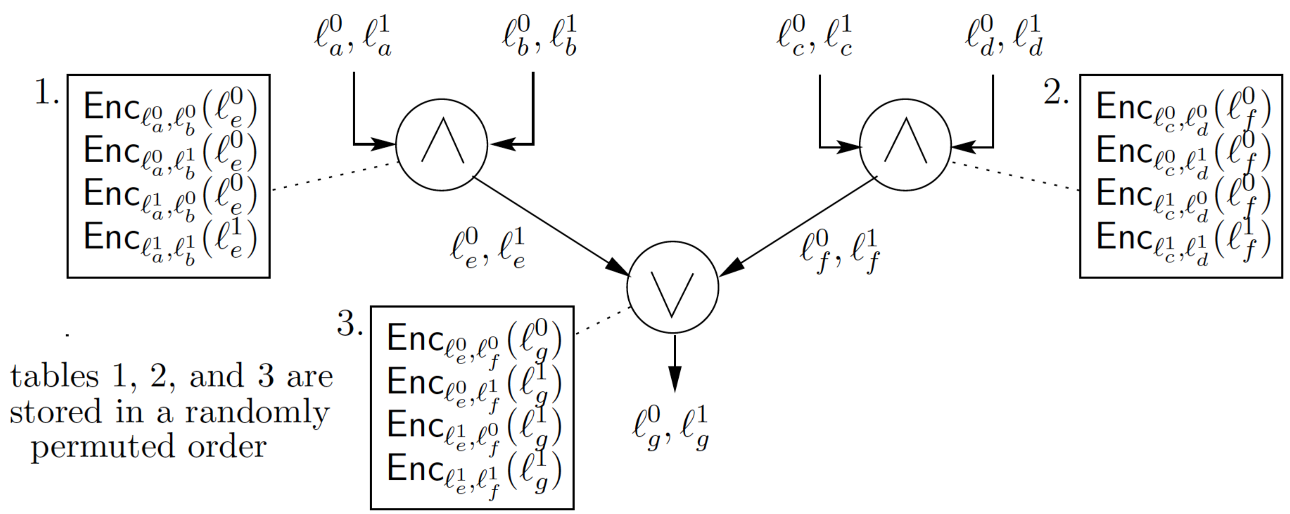
- The garbler takes a Boolean circuit and associates two random labels $\ell_i^0, \ell_i^1 \in \{0, 1\}^\kappa$ with each circuit's wire i
  - ℓ_i^0 is associated with value 0 of the wire and ℓ_i^1 with value 1
  - given ℓ_i^b, it is not possible to determine what b is



[Y86] A. Yao, "How to generate and exchange secrets," 1986.

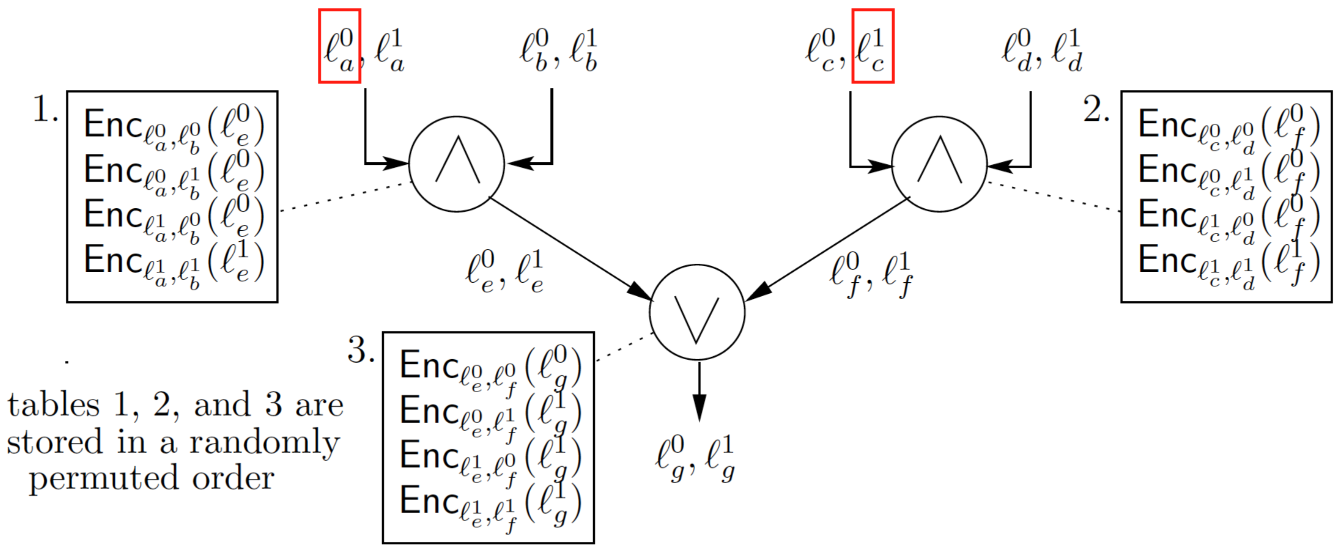
Yao's Protocol: Garbled Circuit Evaluation

- The garbler also encodes each gate and sends it to the evaluator
  - suppose a binary gate g has input wires i and j and output wire k
  - the garbler uses encryption to enable recovery of $\ell_k^{g(b_i,b_j)}$ given $\ell_i^{b_i}$ and $\ell_j^{b_j}$



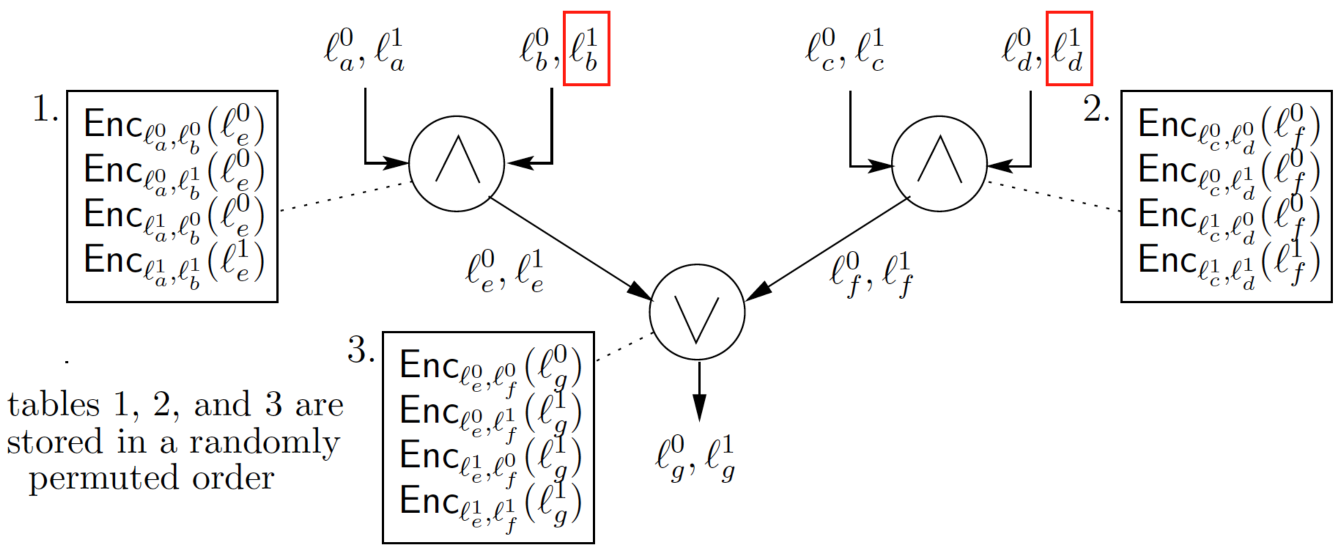
Yao's Protocol: Garbled Circuit Evaluation

- The garbler sends the label corresponding to its own input bit
  - the labels are random, so the evaluator does not learn what this bit is



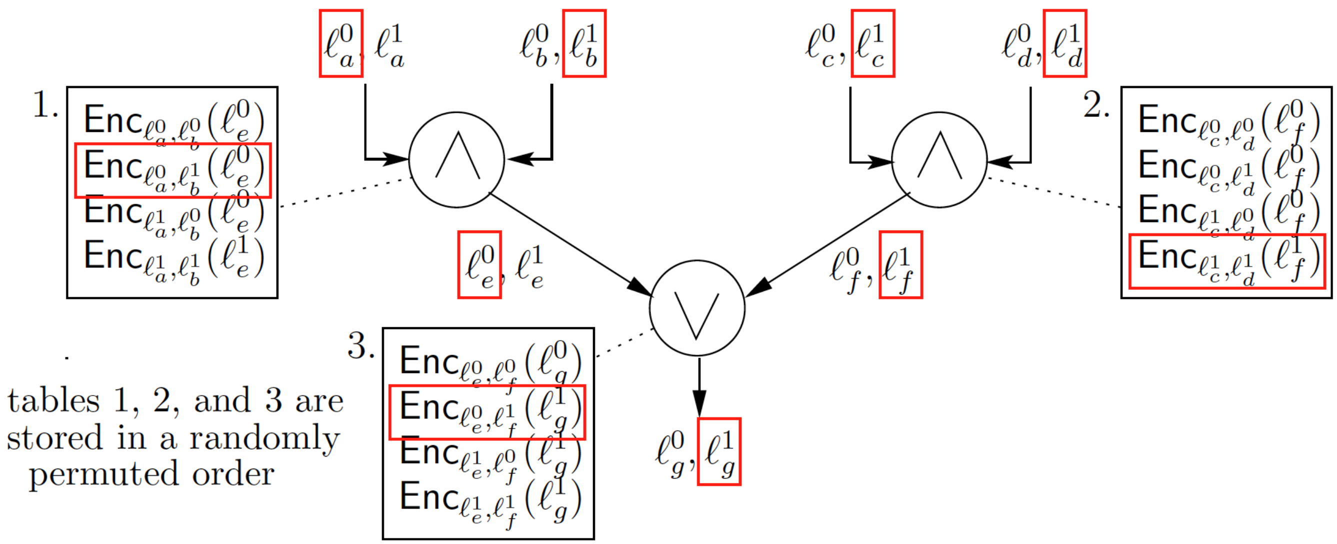
Yao's Protocol: Garbled Circuit Evaluation

- The evaluator engages in 1-out-of-2 oblivious transfer (OT) with the garbler to obtain labels corresponding to its own input
  - it allows the evaluator to retrieve one out of two labels for each of its input wires, while the garbler learns nothing



Yao's Protocol: Garbled Circuit Evaluation

- The evaluator obtains appropriate labels for the input wires and evaluates the garbled circuit one gate at a time
  - the evaluator sees labels, but doesn't know their meaning



Yao's Protocol: Garbled Circuit Evaluation

- At the end of the protocol execution, both parties, one of them, or an external party can learn the output of the protocol execution
- Yao's construction gives a constant-round protocol for secure computation of any function in the semi-honest model
  - the number of rounds does not depend on the number of inputs or the size of the circuit
- The basic technique is secure in the presence of semi-honest garbler and malicious evaluator
  - it can be extended to be secure in the malicious model using additional techniques

Oblivious Transfer

- Oblivious Transfer is a secure two-party protocol, in which the sender holds a number of inputs and the receiver's obtains one of them based on its choice
  - it is used extensively in garbled circuit evaluation
    - at least one OT per input bit, typically an efficiency bottleneck
  - it is also a common tool in other protocols
- Here we are interested in 1-out-of-2 OT, with the sender holding two inputs a_0 and a_1 and the sender holding a bit b
- OT extension allows m (1-out-of-2) OTs to be realized using a constant number of regular OT protocols with small additional overhead linear in m

Oblivious Transfer

- The literature contains many realizations of OT and OT extensions including [NP01, IKNP03, ALSZ13, ALSZ15]

[NP01] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," 2001.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, E. Petrank, "Extending oblivious transfers efficiently," 2003.

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," 2013.

[ALSZ15] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer extensions with security for malicious adversaries," 2015.

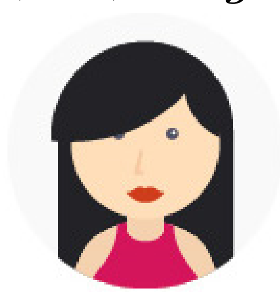
Naor-Pinkas OT

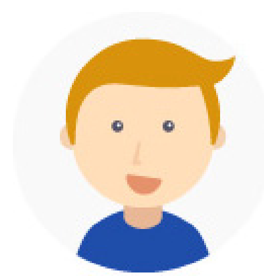
- Naor-Pinkas OT [NP01] is an efficient construction secure in the malicious model
  - sender S inputs two strings ℓ_0 and ℓ_1 and receiver R inputs a bit b
  - common input consists of group $\mathbb{G}$ of prime order q, its generator g, and a random element C of $\mathbb{G}$ (chosen by S)
  - after the protocol, R learns ℓ_b and S learns nothing

Naor-Pinkas OT

- S chooses random $r \in \mathbb{Z}_q$ and computes C^r and g^r

r, C^r, and g^r

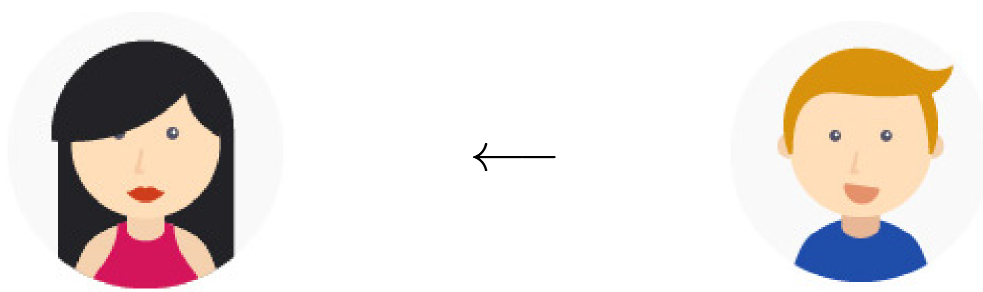




Naor-Pinkas OT

- Receiver R:
  - chooses random $k \in \mathbb{Z}_q^*$
  - sets public keys $PK_b = g^k$ and $PK_{1-b} = C/PK_b$
  - sends PK_0 to S

k, PK_b, and PK_{1-b}

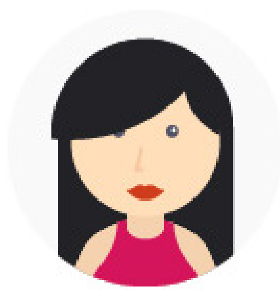


PK_0

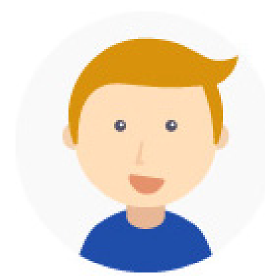
Naor-Pinkas OT

- Consequently, sender S
  - computes $(PK_0)^r$ and $(PK_1)^r = C^r/(PK_0)^r$
  - sends to R g^r and two encryptions $E_0 = H((PK_0)^r, 0) \oplus \ell_0$ and $E_1 = H((PK_1)^r, 1) \oplus \ell_1$
  - here H is a hash function (modeled as a random oracle)

$(PK_0)^r$, $(PK_1)^r$, E_0, and E_1



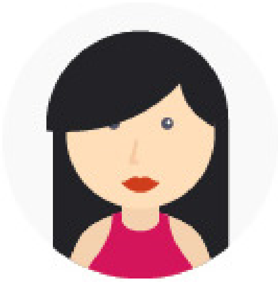
$\longrightarrow$

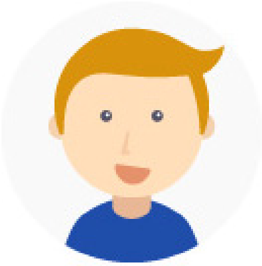


g^r, E_0, and E_1

Naor-Pinkas OT

- R computes $H((g^r)^k) = H((PK_b)^r)$ and uses it to recover ℓ_b

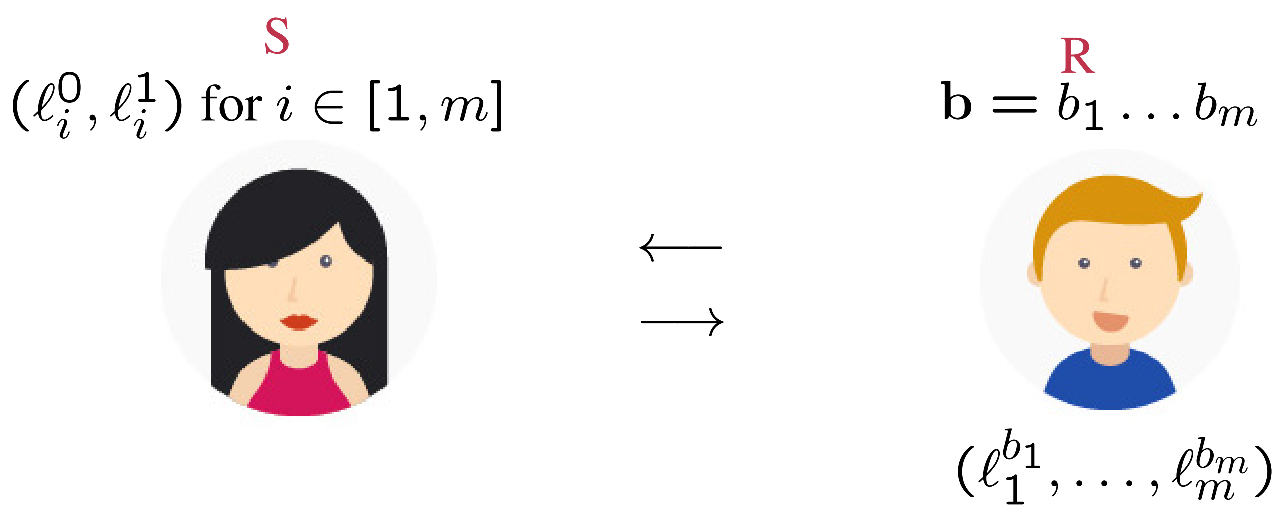




ℓ_b

ALSZ13 OT Extension (Semi-Honest)

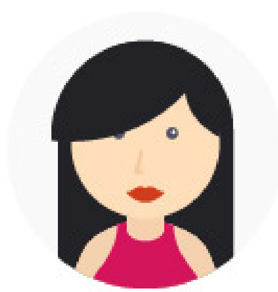
- Asharov-Lindell-Schneider-Zohner OT extension trades public-key operations for symmetric-key operations and communication
- Let sender S hold private binary strings (ℓ_i^0, ℓ_i^1) for $i \in [1, m]$ and receiver R hold m private bits $\mathbf{b} = b_1 \ldots b_m$
- As output, R receives $(\ell_1^{b_1}, \ldots, \ell_m^{b_m})$ and S learns nothing

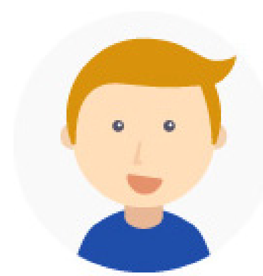


ALSZ13 OT Extension (Semi-Honest)

- S chooses a random string $\mathbf{s} = s_1 \ldots s_\kappa \in \{0, 1\}^\kappa$, where κ is a symmetric-key security parameter

$\mathbf{s}$

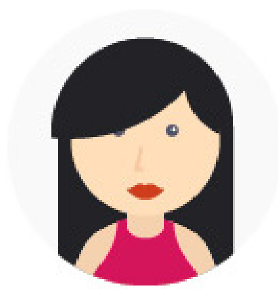


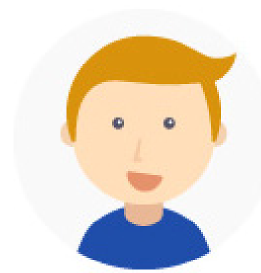


ALSZ13 OT Extension (Semi-Honest)

- R chooses κ pairs of random κ-bit strings (k_i^0, k_i^1) for $i = 1, \ldots, \kappa$

(k_i^0, k_i^1) for $i = 1, \ldots, \kappa$

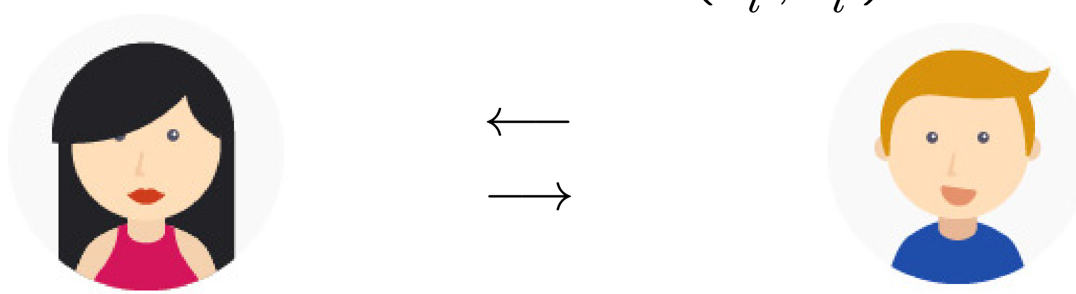




ALSZ13 OT Extension (Semi-Honest)

- S and R perform κ OTs secure against semi-honest parties, with their roles reversed
  - R enters (k_i^0, k_i^1) into the ith OT
  - S inputs s_i
  - S learns $k_i^{s_i}$

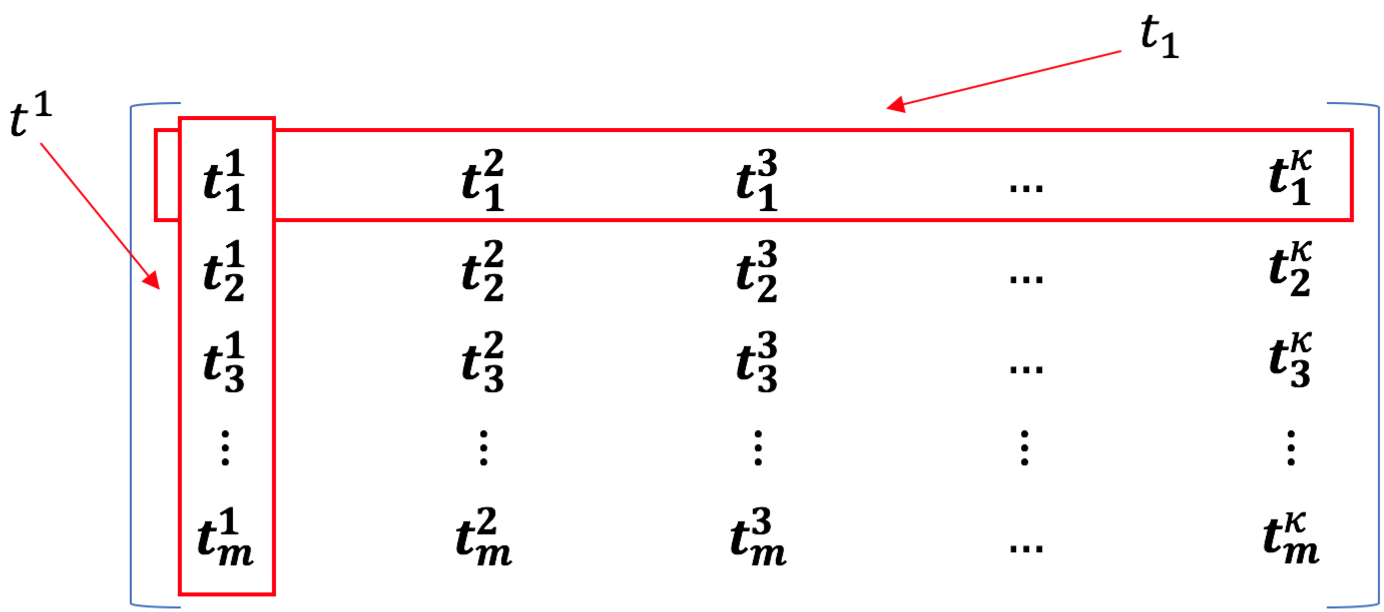
S $\qquad\qquad$ (k_i^0, k_i^1) for $i = 1, \ldots, \kappa$



$(k_1^{s_1}, \ldots, k_\kappa^{s_\kappa})$

ALSZ13 OT Extension (Semi-Honest)

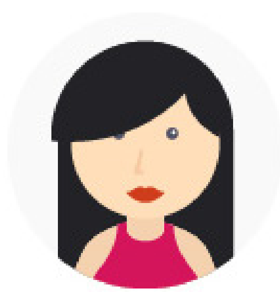
- Let $\mathbf{t}^i = \mathsf{PRG}(k_i^0)$ for $i = 1, \ldots, \kappa$ and $\mathsf{PRG} : \{0,1\}^\kappa \rightarrow \{0,1\}^m$
- Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\kappa]$ denote the $m \times \kappa$ matrix with its ith column being $\mathbf{t}^i$ and jth row being $\mathbf{t}_j$



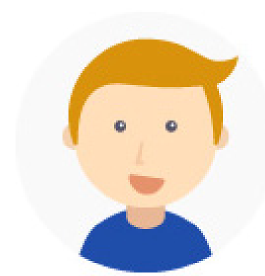
ALSZ13 OT Extension (Semi-Honest)

- R computes $\mathbf{t}^i = \mathsf{PRG}(k_i^0)$, $\mathbf{u}^i = \mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_i^1) \oplus \mathbf{b}$ for $i = 1, \ldots, \kappa$ and sends each $\mathbf{u}^i$ to S

$\mathbf{t}^i$ and $\mathbf{u}^i$ for $i = 1, \ldots, \kappa$



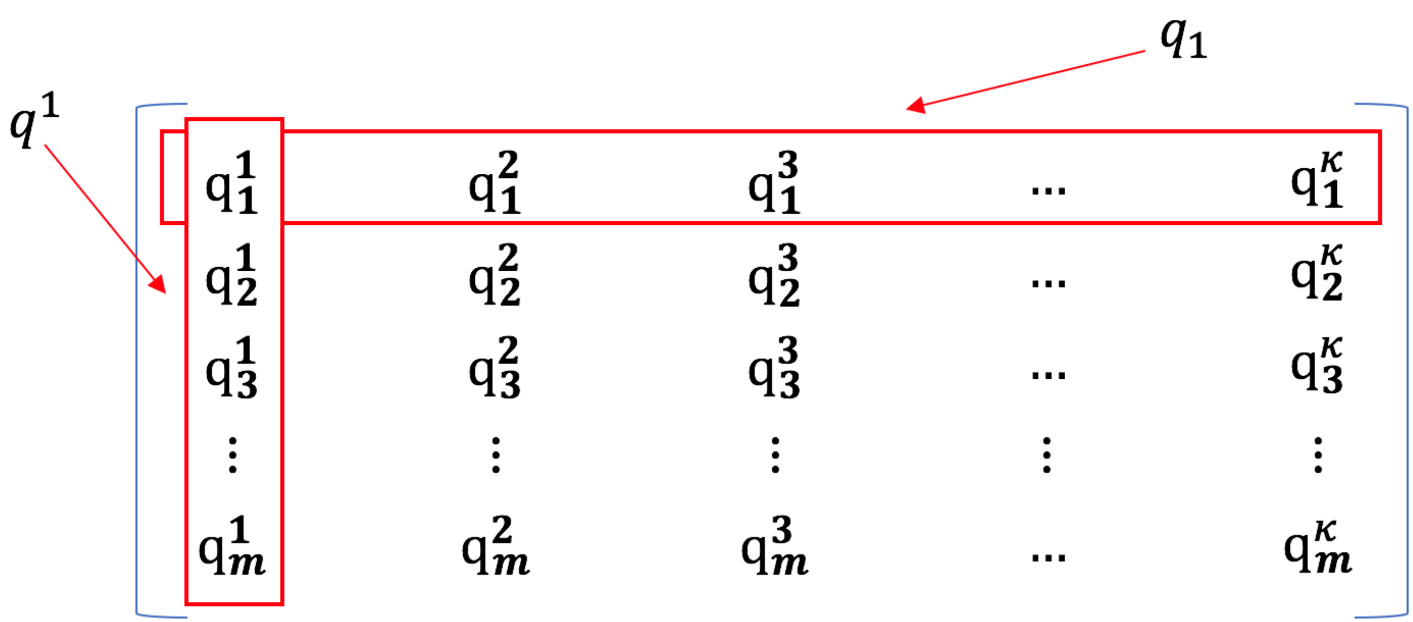
$\longleftarrow$



$\mathbf{u}^i$ for $i = 1, \ldots, \kappa$

ALSZ13 OT Extension (Semi-Honest)

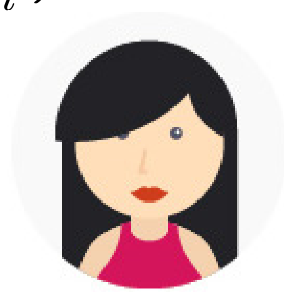
- S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus \mathsf{PRG}(k_i^{s_i}) = (s_i \cdot \mathbf{b}) \oplus \mathbf{t}^i$ for $i = 1, \ldots, \kappa$
- Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\kappa]$ denote the $m \times \kappa$ matrix with its ith column being $\mathbf{q}^i$ and jth row being $\mathbf{q}_j$ where $i = 1, \ldots, \tau$ and $j = 1, \ldots, m$
  - i.e., $\mathbf{q}^i = (s_i \cdot \mathbf{b}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (b_j \cdot \mathbf{s}) \oplus t_j$



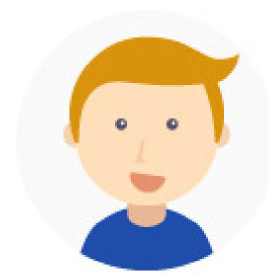
ALSZ13 OT Extension (Semi-Honest)

- S sends to R (w_i^0, w_i^1) for $i = 1, \ldots, m$, where $w_i^0 = \ell_i^0 \oplus H(i, \mathbf{q}_i)$ and $w_i^1 = \ell_i^1 \oplus H(i, \mathbf{q}_i \oplus \mathbf{s})$

(w_i^0, w_i^1) for $i = 1, \ldots, m$



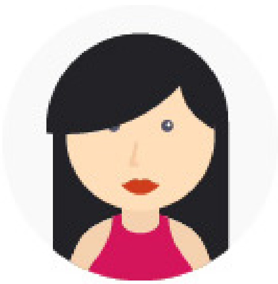
$\longrightarrow$



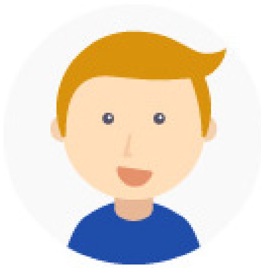
(w_i^0, w_i^1) for $i = 1, \ldots, m$

ALSZ13 OT Extension (Semi-Honest)

- R computes $\ell_i^{b_i} = w_i^{b_i} \oplus H(i, \mathrm{t}_i)$ for $i = 1, \ldots, m$



$(\ell_1^{b_1}, \ldots, \ell_m^{b_m})$



ALSZ15 OT Extension (Malicious)

- The semi-honest OT extension above can be made secure in the presence of malicious adversaries with a few changes:
  - R chooses sets $\mathbf{b}' = \mathbf{b}||\mathbf{r}$ for a random $\mathbf{r} \in \{0, 1\}^\kappa$ and uses $\mathbf{b}'$ in place of $\mathbf{b}$
  - $\mathbf{s}$ is of size $\tau = \kappa + \rho$, where ρ is a statistical security parameter
  - this changes the number of based OTs from κ to τ and matrix dimensions from $m \times \kappa$ to $(m + \kappa) \times \tau$
  - consistency check is required to enforce that the same $\mathbf{b}'$ is used to form each $\mathbf{u}^i$

ALSZ15 OT Extension (Malicious)

- Consistency check cross-checks information about each u^i against u^j's information for each (i, j) pair
  - for every pair $(i, j) \in [1, \tau]^2$, R computes four values:

$$h_{(i,j)}^{(0,0)} = H(\mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_j^0)), \;\; h_{(i,j)}^{(0,1)} = H(\mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_j^1))$$

$$h_{(i,j)}^{(1,0)} = H(\mathsf{PRG}(k_i^1) \oplus \mathsf{PRG}(k_j^0)), \;\; h_{(i,j)}^{(1,1)} = H(\mathsf{PRG}(k_i^1) \oplus \mathsf{PRG}(k_j^1))$$

  and sends them to S

  - for every pair $(i, j) \in [1, \tau]^2$, S checks that
    - $h_{(i,j)}^{(s_i,s_j)} = H(\mathsf{PRG}(k_i^{s_i}) \oplus \mathsf{PRG}(k_j^{s_j}))$
    - $h_{(i,j)}^{(\overline{s}_i,\overline{s}_j)} = H(\mathsf{PRG}(k_i^{s_i}) \oplus \mathsf{PRG}(k_j^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j)$
    - $\mathbf{u}^i \neq \mathbf{u}^j$

Garbled Circuit Evaluation Optimizations

- Multiple optimizations that improve performance of garbled circuit evaluation are known
  - the "free XOR" technique which allows XOR gates to be evaluated very cheaply
  - the garbled row reduction technique which reduces the size of garbled gates
  - the half-gates optimization which further reduces the size of garbled gates
  - performing garbling in a way to permit the use of fixed-key (hardware accelerated) AES which greatly improves the speed of garbling and evaluation

Free XOR

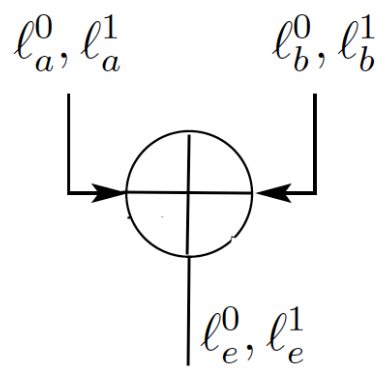
- Garbler has a global secret $\mathbf{R}$ and construct labels as follows:

$$\ell_a^0 \qquad \ell_b^0 \qquad \ell_e^0 = \ell_a^0 \oplus \ell_b^0$$

$$\ell_a^1 = \ell_a^0 \oplus \mathbf{R} \qquad \ell_b^1 = \ell_b^0 \oplus \mathbf{R} \qquad \ell_e^1 = \ell_e^0 \oplus \mathbf{R}$$

$$\ell_a^0 \oplus \ell_b^0 = \ell_a^0 \oplus \ell_b^0 \oplus \mathbf{R} \oplus \mathbf{R} = \ell_a^1 \oplus \ell_b^1$$

$$\ell_a^1 \oplus \ell_b^0 = \ell_a^0 \oplus \ell_b^0 \oplus \mathbf{R} = \ell_a^0 \oplus \ell_b^1$$



- No ciphertexts, encryption, or communication is needed for XOR gates!

  [KS08] V. Kolesnikov and T. Schneider, “Improved garbled circuit: Free XOR gates and applications,” 2008.

Garbled Row Reduction (1)

- The first garbled row reduction optimization reduces the size of a garbled gate from 4 to 3 ciphertexts

- The garbler generates the output labels such that the first entry of the garbled table is derived deterministically and no longer needs to be sent

$$\ell_e^0 = \mathsf{Dec}_{\ell_a^0, \ell_b^0}(0)$$

- This lowers communication, but adds more computational to the garbler side

- It is also compatible with free XOR

[NPS99] M. Naor, B. Pinkas, and R. Sumner. ”Privacy preserving auctions and mechanism design,” 1999.

Garbled Row Reduction (2)

- The second garbled row reduction optimization reduces the size of a garbled gate from 4 to 2 ciphertexts
- The evaluator uses polynomial interpolation over a quadratic curve
- The output label is encoded as the y value on the polynomial at point 0
- As an example for AND gate

$$k_1 = \mathsf{Dec}_{\ell_a^0, \ell_b^0}(0), \quad k_2 = \mathsf{Dec}_{\ell_a^0, \ell_b^1}(0)$$

$$k_3 = \mathsf{Dec}_{\ell_a^1, \ell_b^0}(0), \quad k_4 = \mathsf{Dec}_{\ell_a^1, \ell_b^1}(0)$$

[PSSW09] B. Pinkas, T. Schneider, N. Smart, and S. Williams, “Secure two-party computation is practical,” 2009.

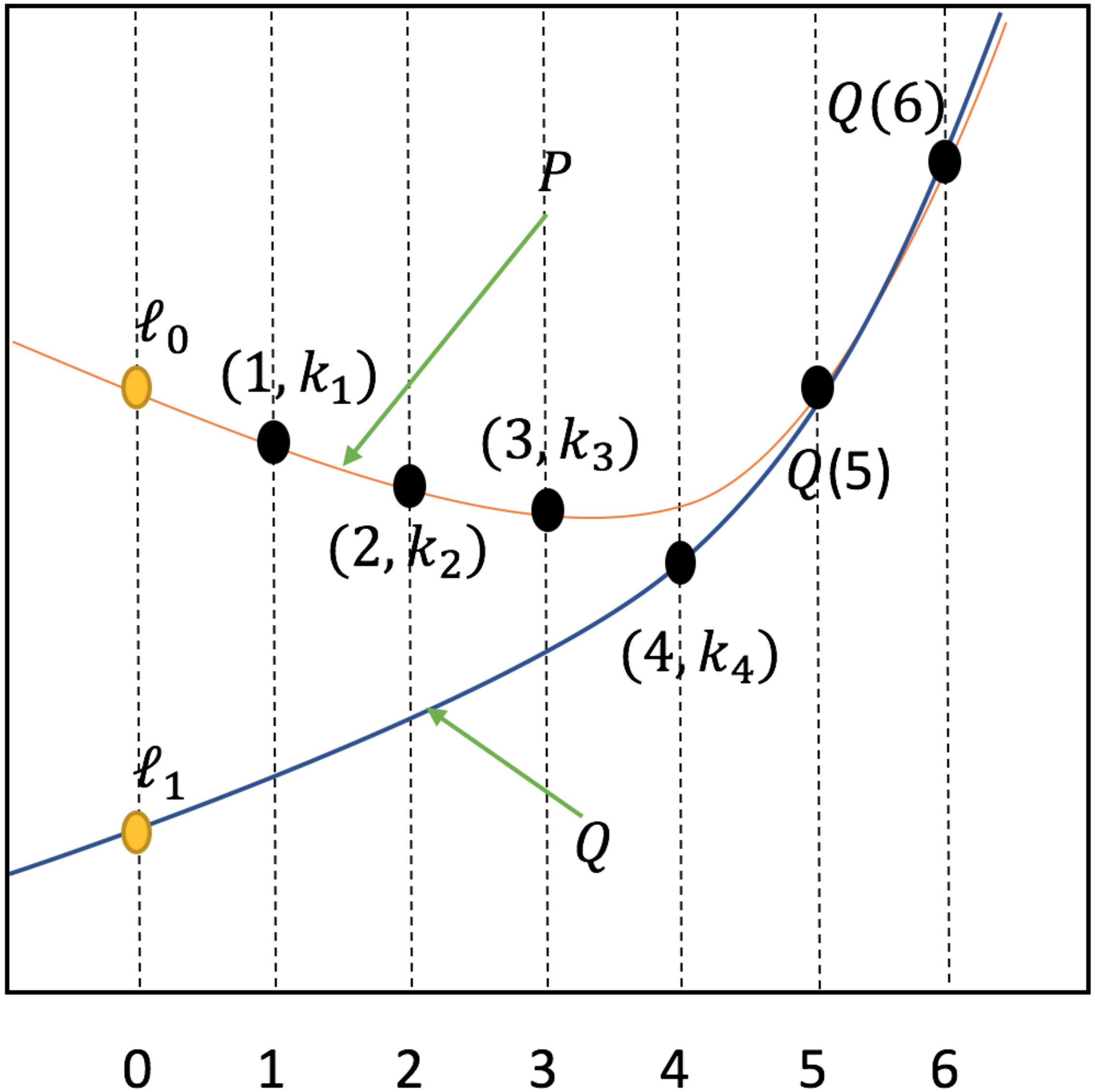
Garbled Row Reduction (2)

- One point on the polynomial is revealed in the usual way and two more (the ones at $x = 5$ and $x = 6$) are included in the garbled gate

- There are two different quadratic polynomials P and Q to consider
  - P and Q are designed to intersect exactly in the two points included in the garbled gate
  - in the Case of AND gate, three points on P are $(\mathsf{Dec}_{\ell_a^0,\ell_b^0}(0), \mathsf{Dec}_{\ell_a^1,\ell_b^0}(0), \mathsf{Dec}_{\ell_a^0,\ell_b^1}(0))$ and three points on Q are $(\mathsf{Dec}_{\ell_a^1,\ell_b^1}(0), Q(5), Q(6))$(with respect to their y-value)

- This is not compatible with free XOR!

Garbled Row Reduction (2)



Half Gates Optimization

- Half-gates is the first optimization technique that simultaneously
  - requires only two ciphertexts per garbled AND gate
  - is compatible with the "free XOR" optimization
- It relies on the fact that

$$a \wedge b = (a \wedge (b \oplus r)) \oplus (a \wedge r)$$

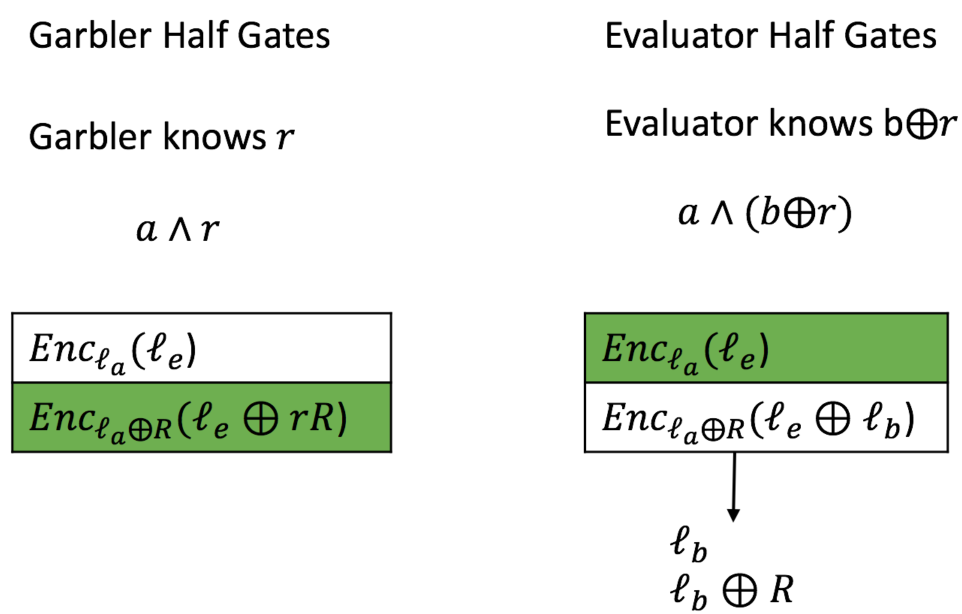
  where r is a random value chosen by the garbler

- The value of $b \oplus r$ is revealed to the evaluator

[ZRE15] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole," 2015.

Half Gates Optimization

- If the green rows are equal to 0 using garbled row reduction, then there are only two ciphertexts are transmit



- Half gates and garbled row reduction techniques reduce bandwidth associated with transmitting garbled gates

Using Fixed-Key Blockcipher

- This optimization modifies how garbled gates are constructed to use fixed-key AES encryption instead of hash functions

- AES hardware implementations are widely available on commodity hardware and allow for significant computation speedup

- This technique is compatible with the “free XOR” and row reduction techniques

[BHKR13] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, “Efficient garbling from a fixed-key blockcipher,” 2013.

Garbled Circuit Evaluation (Malicious)

- Yao's garbled circuit evaluation is not secure in the presence of a malicious garbler
  - there is the need to enforce correct circuit construction and several solutions exist [GMW91], [GMW87], [LP07], [SS11], [L13]
  - we focus on cut-and-choose approaches [LP07], [SS11], [L13]

[GMW91] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems," 1991.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game-or-a completeness theorem for protocols with honest majority," 1987.

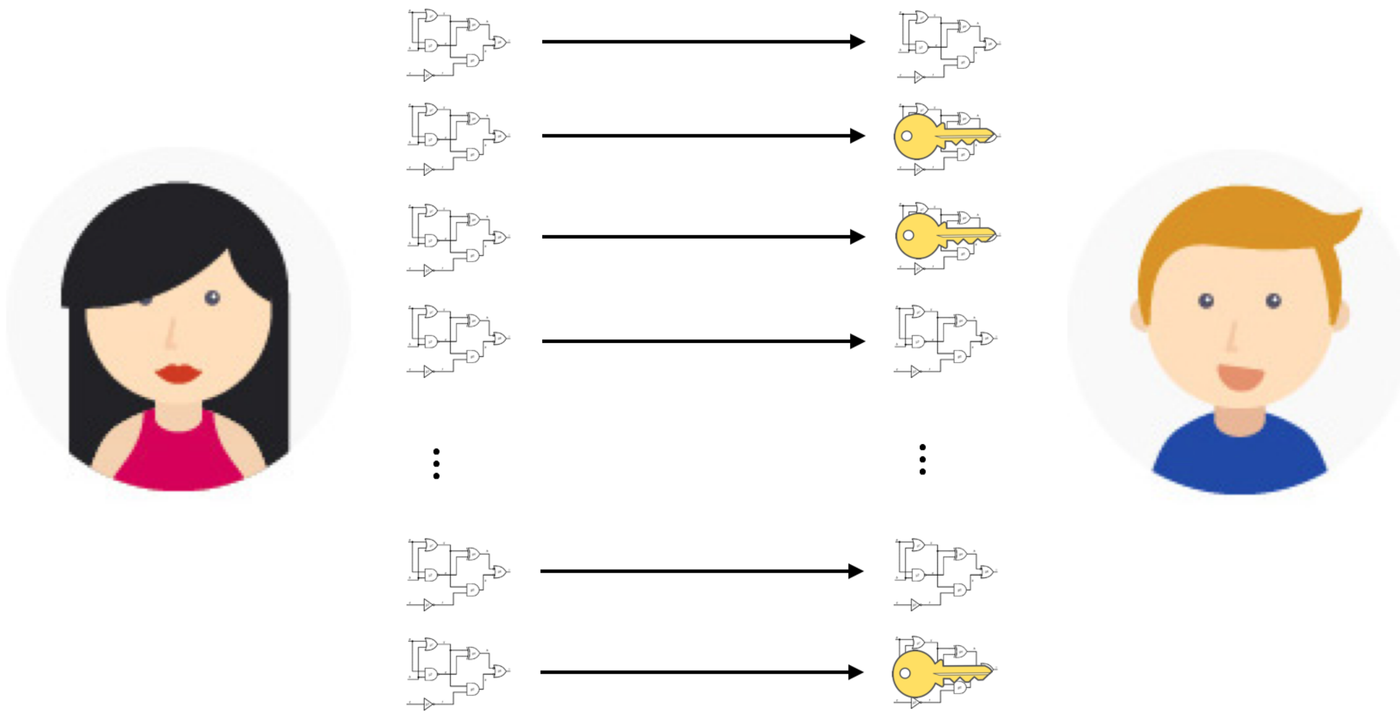
[LP07] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," 2007.

[SS11] A. Shelat and C. Shen, "Two-output secure computation with malicious adversaries," 2011.

[L13] Y. Lindell, "Fast cut-and-choose-based protocols for malicious and covert adversaries," 2013.

Cut-and-Choose

- The garbler generates s independent garblings of a circuit C and opens the circuits of the evaluator's choice



Cut-and-Choose

- The garbler generates s independently garbled versions of circuit C
- The evaluator asks the garbler to open a number of circuits of its choice and garbler reveals the randomness/keys
- The evaluator verifies correctness of the opened circuits
- The parties run OT/OT extension to retrieve the labels corresponding to the evaluator's input for the unopened circuits
- The garbler sends the labels corresponding its own input for the unopened circuits
- The evaluator evaluates the unopened circuits, and returns the majority output

Garbled Circuit Evaluation (Malicious) [LP07]

- Lindell-Pinkas solution proposed the use of cut-and-choose
- By opening a half of the garbled circuits and evaluating the other half, output is incorrect with probability at most $2^{-0.311s}$

Garbled Circuit Evaluation (Malicious) [SS11]

- Shelat-Shen construction used the cut-and-choose approach and proposes novel defence mechanisms for input consistency, selective failure, and output authentication
- It showed that if the garbler opens 60% of the constructed circuits instead 50%, the error decreases from $2^{-0.311s}$ to $2^{-0.32s}$
  - to achieve the error of 2^{-40}, we need approximately 125 circuits instead of 128

Garbled Circuit Evaluation (Malicious) [L13]

- How many circuits needed to be garbled to ensure correct output?
  - previously, for error probability of 2^{-40}, 125 circuits were needed
  - this is a heavy computational overhead compared to the semi-honest solution
- Lindell proposed an optimized cut-and-choose solution that required only s circuits with some small additional overhead to achieve error of 2^{-s}

Garbled Circuit Evaluation (Malicious) [L13]

- Why do we need the majority of the circuits to be correct?
  - an incorrect circuit may compute the desired function if the evaluator's input meets some condition and otherwise compute garbage
  - if the evaluator aborts, it means the garbler knows that the evaluator's input does not meet the condition
  - if the evaluator does not abort, it means the garbler knows that the evaluator's input meets the condition
  - we must enforce that most evaluated circuits are correct with overwhelming probability

Garbled Circuit Evaluation (Malicious) [L13]

- Even if all opened circuits out of s are correct and all unopened circuits are incorrect, the error probability is still bounded by 2^{-s}
- How is it possible?
  - both parties run small additional secure computation
  - if the evaluator receives two different outputs in two different circuits, the additional secure computation allows him to learn the garbler's input
  - in this case, the evaluator can compute the original function f by himself because it knows both inputs
  - the garbler does not know which case happened

Garbled Circuit Evaluation (Malicious)

- The cut-and-choose technique alone does not provide full security
- Additional attacks:
  - input consistency
  - selective failure
  - output authentication

Input Consistency

- When multiple circuits are being evaluated in cut-and-choose, a malicious garbler can provide inconsistent inputs to different evaluation circuits
  - after obtaining the output, the garbler can extract information about the evaluator's input
- Defenses:
  - equality checker [MF06]
  - input commitment [LP07]
  - pseudorandom synthesizer [LP11]
  - malleable claw-free collections [SS11]

[MF06] P. Mohassel and M. Franklin, "Efficiency tradeoffs for malicious two-party computation," 2006.

[LP11] Y. Lindell and B. Pinkas, "Secure two-party computation via cut-and-choose oblivious transfer," 2011.

Selective Failure

- A malicious garbler can also use inconsistent labels during garbling and later during OT
- The evaluator's input can be inferred from whether or not the protocol completes
- Defenses:
  - random input replacement: input bit b is replaced ρ random bits b_i subject to $b = b_1 \oplus b_2 \oplus \ldots \oplus b_\rho$ [LP07]
  - committing OT [K08] [SS11]
  - combining OT and the cut-and-choose steps into one protocol [LP11]

[K08] M. Kiraz, "Secure and fair two-party computation," 2008.

Output Authentication

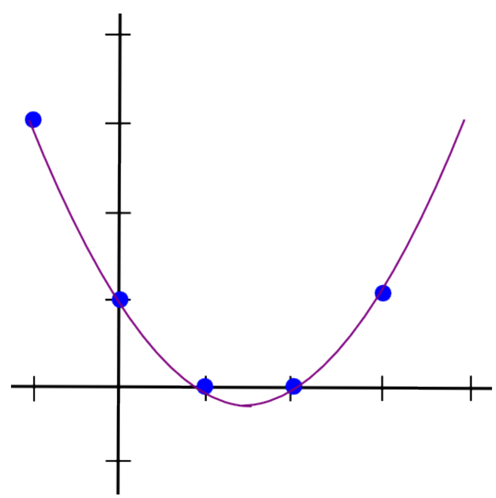
- In many cases, both the garbler and evaluator receive outputs from secure function evaluation, i.e., $f(x, y) = (f_1(x, y), f_2(x, y))$
- A malicious evaluator may claim an arbitrary value to be the generator's output coming from circuit evaluation
- Defenses:
  - verifying authenticity of the garbler's output by modifying the function as $f(x, y) = (f_1(x, y) \oplus c, f_2(x, y))$ and computing its MAC [LP07]
  - using zero knowledge proofs [K08]
  - using a signature-based solution [SS11]

SMC based on Secret Sharing

- An alternative technique is to use threshold linear secret sharing for secure multi-party computation
  - (n, t)-threshold secret sharing allows secret v to be secret-shared among n parties such that:
    - no coalition of t or fewer parties can recover any information about v
    - $t + 1$ or more shares can be used to efficiently reconstruct v
  - information-theoretic security (i.e., independent of security parameters) is achieved

Shamir's (n, t)-Threshold Scheme

- Given n points on the plane $(x_1, y_1), \ldots, (x_n, y_n)$ where all x_is are distinct, there exists an unique polynomial f of degree $\leq n - 1$ such that $f(x_i) = y_i$ for $i = 1, \ldots, n$
  - f can be determined using Lagrange interpolation
- This also holds in a finite field, e.g., in $\mathbb{Z}_p$ where p is prime

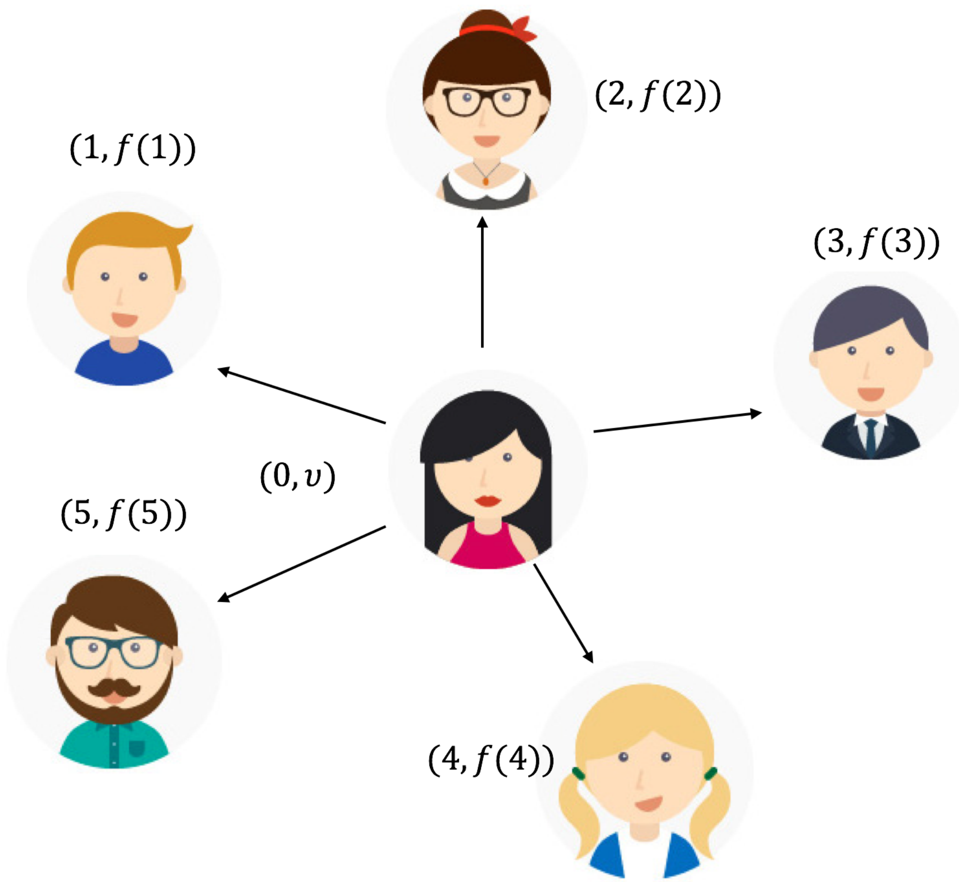


[S79] A,. Shamir, "How to share a secret," 1979.

Shamir's (n, t)-Threshold Scheme

- Shamir secret sharing works as follows
  - suppose we use finite field $\mathbb{Z}_p$ for a prime p
  - choose prime p of sufficient size to represent all values
  - any private value v is represented as an element in Z_p
  - to create shares, choose polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_tx^t \bmod p$, where $a_1, \ldots, a_t$ are random and $a_0 = v$
  - let $[v]$ secret shared v and $[v]_i = (i, f(i))$ represent the share distributed to the ith party for $i \in [1, n]$

Shamir's (n, t)-Threshold Scheme



Shamir's (n, t)-Threshold Scheme

- The secret v can be reconstructed from every subset of $t+1$ or more shares (x_i, y_i) using Langrange interpolation

$$f(x) = \sum_{i=1}^{t+1} y_i \prod_{j=1, j \neq i}^{t+1} \frac{x - x_j}{x_i - x_j} \bmod p$$

$$v = f(0) = \sum_{i=1}^{t+1} y_i \prod_{j=1, j \neq i}^{t+1} \frac{-x_j}{x_i - x_j} \bmod p$$

- Any t or fewer shares do not leak any information about v

SMC based on Shamir Secret Sharing

- Function evaluation is normally expressed using composition of elementary operations
  - functions represented in terms of additions/subtractions and multiplications are called arithmetic circuits
- Performance of any function in this framework is measured in terms of
  - the number of elementary interactive operations
  - the number of sequential interactive operations or rounds

Addition and Subtraction Operations

- Shamir's secret sharing is a linear secret sharing scheme
  - any linear combination of secret shared values can be computed directly on the shares
- Example: addition
  - let $f_1(x) = v_1 + a_1 x + a_2 x^2 + \ldots + a_t x^t$ and $f_2(x) = v_2 + a'_1 x + a'_2 x^2 + \ldots + a'_t x^t$
  - then $g(x) = f_1(x) + f_2(x) = v_1 + v_2 + (a_1 + a'_1)x + (a_2 + a'_2)x^2 + \ldots + (a_t + a'_t)x^t$
  - this means that any party can compute its share of $v_1 + v_2$ as $[v_1]_i + [v_2]_i$ for each i
  - subtraction is performed in a similar way

Multiplication Operation

- Example: scalar multiplication
  - we can multiply secret-shared v by known integer c by directly multiplying each share by c
  - if $f(x) = v + a_1 x + a_2 x^2 + \ldots + a_t x^t$, then $g(x) = c \cdot f(x) = c \cdot v + (c \cdot a_1)x + (c \cdot a_2)x^2 + \ldots + (c \cdot a_t)x^t$
  - $[c \cdot v]_i = c[v]_i$ for each i
- What about multiplication of two secret values?

Multiplication Operation

- To multiply $[v_1]$ and $[v_2]$, each party could locally multiply its shares
  - the product of their representation as $f_1(x)$ and $f_2(x)$ is

$$g(x) = f_1(x) \cdot f_2(x) = v_1 \cdot v_2 + \lambda_1 x + \lambda_2 x^2 + \ldots + \lambda_{2t} X^{2t}$$

  - the polynomials are no longer of degree t, but rather of degree $2t$
  - reduction of the polynomial's degree is needed

Multiplication Operation

- We can write

$$A \cdot \begin{bmatrix} v_1 . v_2 \\ \lambda_1 \\ . \\ . \\ . \\ \lambda_{2t} \end{bmatrix} = \begin{bmatrix} g(0) \\ g(1) \\ . \\ . \\ . \\ g(2t) \end{bmatrix}$$

where A is $(2t+1) \times (2t+1)$ matrix and is defined as $a_{ij} = i^{j-1}$

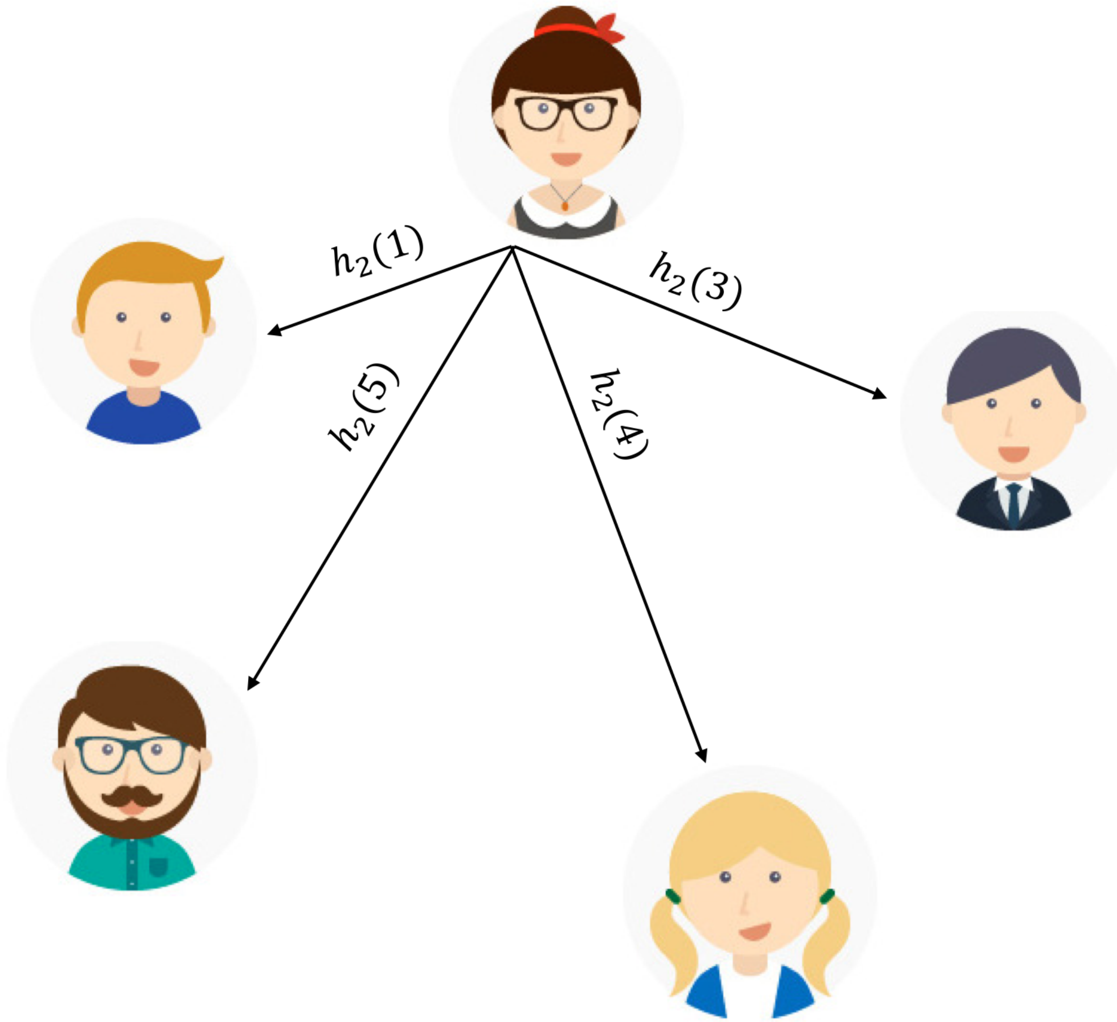
  - A is non-singular and has inverse A^{-1}
  - let the first row of A^{-1} be $[\gamma_0, \gamma_1, \ldots, \gamma_{2t}]$

[GRR98] R. Gennaro, M. Rabin, and T. Rabin, "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography," 1998.

Multiplication Operation

- The inverse equation implies that
  $v_1 \cdot v_2 = g(0)\gamma_0 + g(1)\gamma_1 + \ldots + g(2t)\gamma_{2t}$
- Every player i chooses a random polynomial $h_i(x)$ of degree t such that $h_i(0) = g(i)$
- Let $H(x)$ be defined as $\sum_{i=0}^{2t} \gamma_i h_i(x)$, where $H(0) = v_1 \cdot v_2$
  - this dictates that $2t < n$
- Each player i distributes shares $(j, h_i(j))$ to other players
  - now each player j can compute its own share of $v_1 \cdot v_2$ as $(j, H(j))$
- Polynomial $H(x)$ is of degree t and it is random

Multiplication Operation



SMC based on Secret Sharing

- SMC based on secret sharing supports the flexible setup with three groups of participants:
  - each data owner secret-shares its private input among the computational parties prior to the computation
  - the computational parties evaluate the function on secret-shared data
  - the computational parties communicate their shares of the result to output recipients who locally reconstruct the output

SMC based on Secret Sharing

- A number of techniques are available to strengthen the security guarantees to hold in the malicious model
  - traditionally security has been guaranteed by using verifiable secret sharing techniques
    - each multiplication is followed by a zero-knowledge proof of knowledge that the operation was carried out correctly
    - additional zero-knowledge proofs may be used to prove correct sharing of input or other additional operations
  - more recently computation employs a different structure

Damgård-Nielsen Construction (Malicious)

- Damgård-Nielsen construction works for both semi-honest and malicious models with honest majority
- Multiplication is performed using multiplication triples
  - multiplication triples are of the form (a, b, c) with $c = ab$
  - each of a, b, and c is represented using uniformly random t-sharings
  - triples are generated during the preprocessing phase
  - they are consumed during the online phase

[DN07] I. Damgård and J. Nielsen, "Scalable and unconditionally secure multiparty computation," 2007.

Damgård-Nielsen Construction (Malicious)

- To generate a triple

  1. the parties compute a random value and its two sharings: t-sharing $[r]$ and $2t$-sharing $\langle R \rangle$
  2. all locally parties compute $\langle D \rangle = [a][b] + \langle R \rangle$ on their own shares where shares of random a and b are given
  3. all parties open D which is a uniformly random $2t$-sharing
  4. all parties compute $[c] = D - [r]$ with known D and random t-sharing r (which equals to R)
  5. each party has its own share of (a, b, c)

Damgård-Nielsen Construction (Malicious)

- During online phase, multiplication of secret-shared $[x]$ and $[y]$ is as follows:

  1. choose a fresh triple $[a]$, $[b]$, $[c]$
  2. all parties compute $[\alpha] = [x] + [a]$ and $[\beta] = [y] + [b]$
  3. all parties open α and β
  4. all parties compute $[xy] = -\alpha\beta + \alpha[y] + \beta[x] - [c]$

Damgård-Nielsen Construction (Malicious)

- Inputs are entered using pre-computed random t-sharings $[r]$ known to one party
  - to enter input x, the input owner computes $\delta = x + r$ and broadcasts δ to others
  - all players compute $[x] = \delta - [r]$
- To make it secure in the presence of malicious parties
  - small portions of the protocol utilize verifiable secret sharing (VSS) for generating random elements
  - conflict resolution algorithm is used to enforce consistent sharings
    - many values are verified in a batch

SMC based on Secret Sharing (Malicious)

- SPDZ is another construction that works for malicious models with up to $n-1$ corrupted parties
  - with no majority, the rules of the game change
  - if at least one party misbehaves or aborts, the computation cannot continue
  - we use $(n, n-1)$ secret sharing
    - party i holds a_i such that $a = a_1 + a_2 + \cdots + a_n$

[DPSZ12] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," 2012.

SPDZ (Malicious)

- SPDZ uses the same idea high-level structure as [DN07]
  - computation is divided into the preprocessing and online phases
  - all the expensive public-key operations are performed during preprocessing
  - the online phase is very efficient
- Multiplication also uses precomputed triples
  - this time they are generated using somewhat homomorphic encryption (SHE)
  - zero-knowledge proofs of plaintext knowledge (ZKPoPKs) are used to ensure that the parties encrypt data as they should using SHE

SPDZ (Malicious)

- Computation proceeds on a different representation
  - each private a is secret-shared as

$$\langle a \rangle = (\delta, (a_1, \ldots, a_n), (\gamma(a)_1, \ldots, \gamma(a)_n))$$

  - here $\gamma(a) = \alpha(a + \delta)$ is a MAC on a
  - α is a global private (secret-shared) value (MAC key)
  - each δ is public
  - each party i holds a_i and $\gamma(a)_i$ and each operation updates both values

SPDZ (Malicious)

- SPDZ online computation
  - inputs are entered using pre-generated random values
  - additions are local
  - multiplications consume multiplication triples and are partially open to verify correctness
  - at the end of the computation, the parties open the MAC key α
  - they verify that the MACs on the output (secret-shared) values match the values
    - compute randomized difference, open it, and check for non-zero values
  - if any issues are detected, abort; otherwise, open the results

SPDZ Followup Work

- SPDZ is attractive because of the strong security guarantees and fast online computation
- A number of improved results followed
  - improvements to the offline phase
  - reusability of the MAC key
  - lightweight protocol for covert adversaries

[DKL+13] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. Smart, "Practical covertly secure MPC for dishonest majorityor: breaking the SPDZ limits," 2013.

Compilers for Secure Two-Party Computation

Compiler	PL	AND gate	BW	Adapted by
Fairplay	Java	30 gates/sec	900Bps	
FastGC	Java	96K gates/sec	2.8MBps	CBMC-GC, PCF, SCVM
ObliVM-GC	Java	670K gates/sec	19.6MBps	ObliVM, GraphSc
GraphSC	Java	580K gates/sec per pair of cores	16MBps per pair of cores	
JustGarble	C AES-NI	11M gates/sec	315MBps	TinyGarble

The table is adapted from ObliVM

JustGarble only provides garbling/evaluation (not an end-to-end system)

Compilers for Secure Multi-Party Computation

Compiler	No. parties	Parallelism	Functionality
Sharemind	3	arrays	non-int arithmetic
VIFF	≥ 3	interactive op	varying precision
PICCO	≥ 3	loops, arrays, and user-specified	non-int arithmetic, varying precision
SPDZ	≥ 3	user-specified	non-int arithmetic, non-arithmetic

- The table is adapted from PICCO

[SPDZ] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara, and H. Tsuchida, "Generalizing the SPDZ Compiler for Other Protocols," 2018.

Summary of SMC Techniques

- The two types of SMC techniques described so far can be used to evaluate any function securely
  - depending on the computation, one might be preferred over the other
- A large number of custom protocols for specific functions also exist
  - example: private set intersection
  - these can combine the above techniques or use custom approaches
  - the goal of custom protocols is to outperform general solutions