Session 2: The Yao and BMR Protocols for Secure Computation

Benny Pinkas
Bar-Ilan University
The Yao and BMR Protocols

• Yao presented the first protocol for secure (two-party) computation
• Yao’s protocol was followed by several protocols for the multi-party setting
  – Goldreich-Micali-Wigderson (GMW)
  – Ben Or-Goldwasser-Wigderson (BGW), Chaum-Crepeau-Damgård (CCD)
• Beaver-Micali-Rogaway (BMR) presented a multi-party protocol using a similar approach to Yao’s, and with only \(O(1)\) communication rounds.
Yao’s Protocol
Yao’s Protocol

• A protocol for general secure two-party computation
  – Constant number of rounds
  – The basic protocol is secure only for semi-honest adversaries
  – Many applications of the methodology beyond secure computation

• General secure computation
  – Can securely compute any functionality
  – Based on a representation of the functionality as a Boolean circuit
Representing functions as Boolean circuits???

• In some cases the circuits are small
  – Adding numbers
  – Comparing numbers
  – Multiplying numbers?
  – Computing AES?
  – Working with indirect addressing (A[i])?

• We can efficiently do secure computation of millions and billions of gates
Basic ideas

• A *plain* circuit is evaluated by
  – Setting values to its input gates
  – For each gate, computing the value of the outgoing wire as a function of the wires going into the gate

• Secure computation:
  – No party should learn the values of any internal wires

• Yao’s protocol
  – A compiler which takes a circuit and transforms it to a circuit which hides all information but the final output
Outline

• Garbled circuit
  – An encrypted circuit together with a pair of keys \((k_0,k_1)\) for every wire so that for any gate, given one key on every input wire:
    • It is possible to compute the key of the corresponding gate output
    • It is impossible to learn anything else

• Tool: oblivious transfer
  – Input: sender has \(x_0,x_1\); receiver has \(b\)
  – Receiver obtains \(x_b\) only
  – Sender learns nothing
A Garbled Circuit

• For the entire circuit, assign independent random values/keys to each wire (key $k_0$ for 0, key $k_1$ for 1)
  – These keys are also called “garbled values”

• Encrypt each gate, so that given one key for each input wire, can compute the appropriate key on the output wire
An AND Gate
An AND Gate with Garbled Values

<table>
<thead>
<tr>
<th>u</th>
<th>v</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_u^0$</td>
<td>$k_v^0$</td>
<td>$k_w^0$</td>
</tr>
<tr>
<td>$k_u^0$</td>
<td>$k_v^1$</td>
<td>$k_w^0$</td>
</tr>
<tr>
<td>$k_u^1$</td>
<td>$k_v^0$</td>
<td>$k_w^0$</td>
</tr>
<tr>
<td>$k_u^1$</td>
<td>$k_v^1$</td>
<td>$k_w^1$</td>
</tr>
</tbody>
</table>
A Garbled AND Gate

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$u^0$</td>
<td>$k^0_w$</td>
<td>$k^1_w$</td>
</tr>
<tr>
<td>$u^1$</td>
<td>$k^1_w$</td>
<td>$k^0_w$</td>
</tr>
<tr>
<td>$v^0$</td>
<td>$k^0_u$</td>
<td>$k^0_v$</td>
</tr>
<tr>
<td>$v^1$</td>
<td>$k^1_u$</td>
<td>$k^1_v$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$u$</th>
<th>$v$</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k^0_u$</td>
<td>$k^0_v$</td>
<td>$E_{k^0_u}(E_{k^0_v}(k^0_w))$</td>
</tr>
<tr>
<td>$k^0_u$</td>
<td>$k^1_v$</td>
<td>$E_{k^0_u}(E_{k^1_v}(k^0_w))$</td>
</tr>
<tr>
<td>$k^1_u$</td>
<td>$k^0_v$</td>
<td>$E_{k^1_u}(E_{k^0_v}(k^0_w))$</td>
</tr>
<tr>
<td>$k^1_u$</td>
<td>$k^1_v$</td>
<td>$E_{k^1_u}(E_{k^1_v}(k^1_w))$</td>
</tr>
</tbody>
</table>
A Garbled AND Gate

• The actual garbled gate

\[
\begin{aligned}
&E_{k_u^1}(E_{k_v^0}(k_{w}^0)) \\
&E_{k_u^0}(E_{k_v^1}(k_{w}^0)) \\
&E_{k_u^1}(E_{k_v^1}(k_{w}^1)) \\
&E_{k_u^0}(E_{k_v^0}(k_{w}^1))
\end{aligned}
\]

• Given \(K_u^0\) and \(K_v^1\) can obtain only \(K_w^0\)
• Furthermore, since the order of the rows is permuted, the party has no idea if it obtained the 0 or 1 key
Output Translation

• If the gate is an output gate, also need to provide the “decryption” of the output wire

• Output translation table:
  
  \[ [(0, k_w^0), (1, k_w^1)] \]

• (Note: this table is insecure if the wire w is used as an input wire to any other gate. It is better to use a table of the form \[ [(0, H(k_w^0)), (1, H(k_w^1))] \] but this complicates the security proof.)
Constructing a Garbled Circuit

• Given a Boolean circuit
  – Assign garbled values to all wires
  – Construct garbled gates using the garbled values

• Central property:
  – Given a garbled value for each input wire, can compute the entire circuit, and obtain garbled values for the output wires
  – Given a translation table for the output wires, can obtain output
  – Nothing but the final output is learned!
An Example Circuit

\[(0, k_f^0), (1, k_f^1)\] \hspace{1cm} \[(0, k_g^0), (1, k_g^1)\]

\[
E_{k_d^0}(E_{k_c^0}(k_f^0)) \\
E_{k_d^0}(E_{k_c^1}(k_f^0)) \\
E_{k_d^1}(E_{k_c^0}(k_f^0)) \\
E_{k_d^1}(E_{k_c^1}(k_f^1)) \\
E_{k_a^0}(E_{k_b^0}(k_c^0)) \\
E_{k_a^0}(E_{k_b^1}(k_c^0)) \\
E_{k_a^1}(E_{k_b^0}(k_c^0)) \\
E_{k_a^1}(E_{k_b^1}(k_c^1)) \\
E_{k_a^1}(E_{k_b^1}(k_c^0)) \\
E_{k_a^1}(E_{k_b^1}(k_c^1))
\]

\[
x_1 \hspace{1cm} x_2 \hspace{1cm} y_1 \hspace{1cm} y_2
\]

Bar-Ilan University, Israel, 2015
Computing a Garbled Circuit

- How does the party computing the circuit know that it decrypted the “correct” entry?
  - A gate table has four entries in permuted order
  - The keys known to the evaluator can decrypt only a single entry, but symmetric encryption may decrypt “correctly” even with incorrect keys

- Two possibilities (actually many...)
  - Add redundant zeroes to the plaintext; only correct keys give redundant block
  - Add a bit to signal which ciphertext to decrypt
Computing a Garbled Circuit

• **Option 1:**
  – Encryption: $E_K(m) = [r, F_K(r) \oplus (m | 0^n)]$
  – By the pseudo-randomness of $F$, the probability of obtaining $0^n$ with an incorrect $K$ is negligible

• **Option 2:**
  – For every wire, choose a random signal bit together with the keys

Each wire has an “internal value” bit which must be kept secret
• It also has an “external value” bit which the evaluator can see
• External value is equal to (internal value xor signal bit)
Computing a Garbled Circuit with a Signal Bit

• The actual garbled gate table ordered based on external values

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>$E_{k^1_u} (E_{k^0_v} (k^0_w \parallel 1))$</td>
</tr>
<tr>
<td>(0,1)</td>
<td>$E_{k^1_u} (E_{k^1_v} (k^1_w \parallel 0))$</td>
</tr>
<tr>
<td>(1,0)</td>
<td>$E_{k^0_u} (E_{k^0_v} (k^0_w \parallel 1))$</td>
</tr>
<tr>
<td>(1,1)</td>
<td>$E_{k^0_u} (E_{k^1_v} (k^0_w \parallel 1))$</td>
</tr>
</tbody>
</table>

• Advantage
  - Evaluator knows external values and therefore which entry to decrypt. Computing the circuit requires just two decryptions per gate (rather than an average of 5 if $0^n$ is appended to plaintext)
Yao’s protocol

• $P_1$ sends to $P_2$
  – Tables encoding each circuit gate.
  – The keys corresponding to $P_1$’s input values.

• If $P_2$ gets the keys corresponding to its input values, it can compute the output of the circuit, and nothing else.
  – Why can’t $P_1$ provide $P_2$ with the keys corresponding to both 0 and 1 for $P_2$’s input wires?
Yao’s protocol

• For every wire $i$ of $P_2$’s input:
  – The parties run an OT protocol
  – $P_2$’s input is her input bit ($y_i$).
  – $P_1$’s input is $k_i^0, k_i^1$
  – $P_2$ learns $k_i^{y_i}$

• The OTs for all input wires can be run in parallel.

• Afterwards $P_1$ can compute the circuit by itself.
Yao’s Protocol

- **Input**: $x$ and $y$ of length $n$
- $P_1$ generates a garbled circuit $G(C)$
  - $k_L^0, k_L^1$ are the keys on wire $w_L$
  - Let $w_1, \ldots, w_n$ be the input wires of $P_1$ and $w_{n+1}, \ldots, w_{2n}$ be the input wires of $P_2$
- $P_1$ sends to $P_2$ $G(C)$ and the strings $k_1^{x_1}, \ldots, k_n^{x_n}$
- $P_1$ and $P_2$ run $n$ OTs in parallel
  - $P_1$ inputs $(k_{n+i}^0, k_{n+i}^1)$
  - $P_2$ inputs $y_i$
- Given all keys, $P_2$ computes $G(C)$ and obtains $C(x, y)$
  - $P_2$ sends result to $P_1$
The Example Circuit

(input wires \( P_1 = d,a; P_2 = b,e \))

\[ [(0, k_f^0), (1, k_f^1)] \quad [(0, k_g^0), (1, k_g^1)] \]
Double-Encryption Security

• Need to formally prove that given 4 encryptions of a garbled gate and only 2 keys
  – Nothing is learned beyond one output
• Actually, in order to simulate the protocol, we need something stronger
• Notation:
  – Double encryption: $\overline{E}(k_u, k_v, m) = E_{k_u}(E_{k_v}(m))$
  – Oracles: $\overline{E}(\cdot, k_v, \cdot), \overline{E}(k_u, \cdot, \cdot)$
Double-Encryption Security

\[
\text{Expt}^{\text{double}}_A(n, \sigma)
\]

1. The adversary \(A\) is invoked upon input \(1^n\) and outputs two keys \(k_0\) and \(k_1\) of length \(n\) and two triples of messages \((x_0, y_0, z_0)\) and \((x_1, y_1, z_1)\) where all messages are of the same length.

2. Two keys \(k'_0, k'_1 \leftarrow G(1^n)\) are chosen for the encryption scheme.

3. \(A\) is given the challenge ciphertext \((E(k_0, k'_1, x_\sigma), E(k'_0, k_1, y_\sigma), E(k'_0, k'_1, z_\sigma))\) as well as oracle access to \(E(\cdot, k'_1, \cdot)\) and \(E(k'_0, \cdot, \cdot)\).

4. \(A\) outputs a bit \(b\) and this is taken as the output of the experiment.

Enabling \(A\) to access these oracles gives it more power

The encryption is secure if the adversary cannot identify which one of the two triples as encrypted
Proof of Security – $P_1$ is Corrupted

- $P_1$’s view consists only of the messages it receives in the oblivious transfers
- In the OT-hybrid model, $P_1$ receives no messages in the oblivious transfers
- Simulation:
  - Generate an empty transcript
Proof of Security – P_2 is Corrupted

• More difficult case
  – Need to construct a fake garbled circuit \( G(C') \) that looks indistinguishable to \( G(C) \)
  – Simulated view contains keys to input wires and \( G(C') \)
  – \( G(C') \) together with the keys computes \( f(x,y) \)
  – But the simulator does not know \( x \), so cannot generate a real garbled circuit
Proof of Security – $P_2$ is Corrupted

• The simulator
  – Given $y$ and $z = f(x,y)$, construct a fake garbled circuit $G'(C)$ that always outputs $z$
    • Do this by choosing wire keys as usual, but encrypting the same output key in all ciphertexts, e.g.
      $E_{k_u^1}(E_{k_v^0}(k_w^0))$ $E_{k_u^1}(E_{k_v^1}(k_w^0))$
      $E_{k_u^0}(E_{k_v^1}(k_w^0))$ $E_{k_u^0}(E_{k_v^0}(k_w^0))$
    • This ensures that no matter the input, the same known garbled values on the output wires are received
Proof of Security – \( P_2 \) is Corrupted

• Simulator (continued)
  – Simulation of output translation tables
    • Let \( k,k' \) be the keys on the \( i^{\text{th}} \) output wire; let \( k \) be the key encrypted in all 4 entries of the gate which outputs this wire
    • If \( z_i = 0 \), write \([ (0,k),(1,k') ]\]
    • If \( z_i = 1 \), write \([ (0,k'),(1,k) ]\]
  – Simulation of input keys phase
    • Input wires associated with \( P_1 \)'s input: send any one of the two keys on the wire
    • Input wires associated with \( P_2 \)'s input: simulate output of OT to be any one of the two keys on the wire
Proof of Security – $P_2$ is Corrupted

• Need to prove that the simulation is indistinguishable from the real execution

• First step – modify simulator as follows
  – Given circuit inputs $x$ and $y$ (just for the sake of the proof), label all keys on the wires as **active** or **inactive**
    • **active**: key is obtained on this wire upon inputs $(x, y)$
    • **inactive**: key is not obtained on wire upon inputs $(x, y)$
  – Make sure that the single key encrypted in each gate is the **active** one

• This simulation is identical to the previous one
Proof of Security – P₂ is Corrupted

• Proven by a hybrid argument
  – Consider a garbled circuit $G_L(C)$ for which:
    • The first $L$ gates are generated as in the (alternative) simulation
    • The rest of the gates are generated honestly
  • Claim: $G_{L-1}(C)$ is indistinguishable from $G_L(C)$

  • Proof:
    – Difference is in $L^{th}$ gate
    – Intuition: use indistinguishability of encryptions to say that cannot distinguish real garbled gate from one where the same active key is encrypted in all entries
Proof of Security – $P_2$ Corrupted

• Observation – $L^{th}$ gate
  – The encryption under both active keys is identical in both cases
  – The difference is encryptions where one or both of the keys are inactive keys
    • Must show that these three encryptions are indistinguishable from the encryptions in real execution

• The problem
  – The inactive keys in this gate may appear in other gates as well
    • We needed the oracles to generate these other encryptions...
The Example Circuit

(input wires $P_1 = d,a; P_2 = b,e$)

\[
[(0, k_f^0), (1, k_f^1)] \quad [(0, k_g^0), (1, k_g^1)]
\]
Simulator’s Circuit (Output 01)

\[ [(0, k_f^0), (1, k_f^1)] \quad [(0, k_g^0), (1, k_g^1)] \]

In each gate, all table entries are identical.
Inactive Keys

Assuming input is (da=01, be=10), output is (fg=01)

\[ [(0, k_f^0), (1, k_f^1)] \quad [(0, k_g^0), (1, k_g^1)] \]
Inactive Keys

Assuming input is \((da=01, be=10)\), output is \((fg=01)\)

\[
[(0, k_f^0), (1, k_f^1)] \quad [(0, k_g^0), (1, k_g^1)]
\]
Modify Simulator
(Encrypt Active Keys Only)

\[(0, k_f^0), (1, k_f^1)\] \quad \[(0, k_g^0), (1, k_g^1)\]

\begin{align*}
E_{k_d^0} (E_{k_c^0} (k_f^0)) \\
E_{k_d^0} (E_{k_c^1} (k_f^0)) \\
E_{k_d^1} (E_{k_c^0} (k_f^0)) \\
E_{k_d^1} (E_{k_c^1} (k_f^0))
\end{align*}

\begin{align*}
E_{k_d^0} (E_{k_c^0} (k_f^1)) \\
E_{k_d^0} (E_{k_c^1} (k_f^1)) \\
E_{k_d^1} (E_{k_c^0} (k_f^1)) \\
E_{k_d^1} (E_{k_c^1} (k_f^1))
\end{align*}

\begin{align*}
E_{k_e^0} (E_{k_c^0} (k_g^0)) \\
E_{k_e^0} (E_{k_c^1} (k_g^0)) \\
E_{k_e^0} (E_{k_c^0} (k_g^1)) \\
E_{k_e^0} (E_{k_c^1} (k_g^1))
\end{align*}

\begin{align*}
E_{k_e^1} (E_{k_c^0} (k_g^0)) \\
E_{k_e^1} (E_{k_c^1} (k_g^0)) \\
E_{k_e^1} (E_{k_c^0} (k_g^1)) \\
E_{k_e^1} (E_{k_c^1} (k_g^1))
\end{align*}

Note change in encrypted key
Hybrid on OR Gate – Simulated OR

\[
\begin{align*}
(0, k_f^0), (1, k_f^1) & \quad \text{REAL} \\
(0, k_g^0), (1, k_g^1) & \quad \text{SIM}
\end{align*}
\]
Hybrid on OR Gate – Real OR

\[
\left(0, k_0^f \right), \left(1, k_1^f \right) \quad \left(0, k_0^g \right), \left(1, k_1^g \right)
\]

REAL

SIM

Secure Computation and Efficiency
Bar-Ilan University, Israel, 2015
What’s the Difference

• In the simulated OR case, the inactive key $k_c^0$ encrypts the key $k_g^1$

• In the real OR case, the inactive key $k_c^0$ encrypts the key $k_g^0$

• Indistinguishability follows from the indistinguishability of encryptions under the inactive key $k_c^0$
Proving Indistinguishability

• Follows from the indistinguishability of encryptions under the inactive key $k_c^0$

• The good news
  – Key $k_c^0$ is not encrypted anywhere (as data) because prior gates are simulated

• The bad news
  – The key $k_c^0$ needs to be used to construct the real AND gate for the hybrid

• The solution
  – The special double-encryption CPA game
The problem: inactive key used in another gate

\[(0, k^0_f), (1, k^1_f)\]  \[\quad(0, k^0_g), (1, k^1_g)\]
Double-Encryption Security

\[ \text{Expt}^{\text{double}}_A (n, \sigma) \]

1. The adversary \( A \) is invoked upon input \( 1^n \) and outputs two keys \( k_0 \) and \( k_1 \) of length \( n \) and two triples of messages \((x_0, y_0, z_0)\) and \((x_1, y_1, z_1)\) where all messages are of the same length.

2. Two keys \( k'_0, k'_1 \leftarrow G(1^n) \) are chosen for the encryption scheme.

3. \( A \) is given the challenge ciphertext \( \langle E(k_0, k'_1, x_\sigma), E(k'_0, k_1, y_\sigma), E(k'_0, k'_1, z_\sigma) \rangle \) as well as oracle access to \( E(\cdot, k'_1, \cdot) \) and \( E(k'_0, \cdot, \cdot) \).

4. \( A \) outputs a bit \( b \) and this is taken as the output of the experiment.

- \( k_0, k_1 \) (i.e., \( k_c^1, k_e^0 \)) are active keys
- \( k'_0, k'_1 \) (i.e., \( k_c^0, k_e^1 \)) are inactive keys
  - Can use oracle to generate the REAL AND gate
Proof of Security – $P_2$ is Corrupted

• Since each gate-replacement is indistinguishable, using a hybrid argument we have that the distributions are indistinguishable (see paper for details)

• QED
Efficiency

• 2-4 rounds (depending on OT and if one party or both parties receive output)
• $|y|$ oblivious transfers
• $8|C|$ symmetric encryptions to generate circuit and $2|C|$ to compute it (using the signal bit)
• For a circuit of 33,000 gates, about 528 Kbytes with 128bit AES encryption
Malicious Adversaries

• Assume that the OT is secure for malicious adv:
  – A corrupted \( P_1 \) cannot learn anything (it receives no messages in the protocol, in the hybrid-OT model)
    • Thus, we have privacy
  – We can prove full security for the case of a corrupted \( P_2 \)

• This can be useful, but...
  – This does not ensure that the parties compute the required functionality
  – E.g., consider \( P_1 \) that builds circuit so that if \( P_2 \)’s first bit is 0, the circuit doesn’t decrypt
    • If \( P_1 \) can detect this in the real world, privacy is lost
  – Proving full security against a malicious \( P_1 \) is hard
The BMR Protocol
The BMR protocol

- Beaver-Micali-Rogaway
- A multi-party version of Yao’s protocol
- Works in $O(1)$ communication rounds, regardless of the depth of the Boolean circuit. (The GMW,BGW, CCD protocol have $O(d)$ rounds)

The BMR protocol: the basic idea

• Two random seeds (aka keys, garbled values) are set for every wire of the Boolean circuit:
  – Each seed is a concatenation of seeds generated by all players and secretly shared among them.

• The parties securely compute together a 4x1 table for every gate (in parallel):
  – Given a 0/1 seed to each of the two input wires, the table reveals the seed of the resulting value of the output wire.
Encoding Gates

- Wire $a$ has seeds $s_{a,1}^0, s_{a,1}^1, \ldots, s_{a,n}^0, s_{a,n}^1$ of parties $P_1, \ldots, P_n$.
- Every wire has similar seeds.
- Each wire has a secret bit $\lambda$. If $\lambda_a = 0$ then $s_{a,i}^0$ corresponds to an internal value of 0 and $s_{a,i}^1$ corresponds to an internal value of 1. Otherwise $s_{a,i}^0$ corresponds to 1 and $s_{a,i}^1$ to 0.
- The $\lambda$ values are random and shared between the parties, so no one knows to which internal value the 0 seeds correspond.
Encoding Gates

• Suppose that $\lambda_a = 0$, $\lambda_b = 1$ and $\lambda_c = 0$.

• The seeds $s_{a,1}^0, \ldots, s_{a,n}^0$ and $s_{b,1}^0, \ldots, s_{b,n}^0$
  – Correspond to internal values of $a = 0$, $b = 1$, and consequently to $c = 0$.
  – Since $\lambda_c = 0$ they will encrypt the corresponding seeds of wire $c$, $s_{c,1}^0, \ldots, s_{c,n}^0$

• Can similarly decide which seed of wire $c$ must be encrypted by each combination of the seeds of wires $a, b$. 
Encoding Gates

• For each gate, the table encrypting the outputs of the gate is a function of
  – $\lambda_a=0$, $\lambda_b=1$, $\lambda_c=0$ (these values are shared by the parties)
  – The seeds $s_{a,1}^0, s_{a,1}^1, \ldots, s_{a,n}^0, s_{a,n}^1, s_{b,1}^0, s_{b,1}^1, \ldots, s_{b,n}^0, s_{b,n}^1$
  – $s_{c,1}^0, s_{c,1}^1, \ldots, s_{c,n}^0, s_{c,n}^1$
  – Gate type (AND, OR, etc.)

• The size of this function is independent of the circuit size

• The parties can run a secure computation to compute the table (using, e.g., GMW etc.)
The BMR protocol

• Offline: The parties securely compute together a 4x1 table for every gate (in parallel for all gates):
  – This is essentially a secure computation of the table
  – All tables are computed in parallel. Therefore overall O(1) rounds.
  – This is the main bottleneck of the BMR protocol (FairplayMP optimizes this computation).

• Online: Given the tables and the seeds of the input values, compute the circuit as in Yao.
Summary

• Can compute any functionality securely in presence of semi-honest adversaries.

• The Yao and BMR protocols are efficient, for circuits that are not too large.

• Obtaining security against malicious adversaries is hard.