CRYPTOGRAPHIC PROTOCOLS
2016, LECTURE 10

Garbled Circuits

HELGER LIPMAA, UNIVERSITY OF TARTU
UP TO NOW

- We saw secure computation protocols
- with tradeoffs

<table>
<thead>
<tr>
<th></th>
<th>Rounds</th>
<th>Communication</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDD-based</td>
<td>🟢🟢🟢</td>
<td>🟢🟢🟢</td>
<td>🔴🔴🔴</td>
</tr>
<tr>
<td>MPC</td>
<td>🔴🔴🔴</td>
<td>🔴🔴🔴</td>
<td>🟢🟢🟢</td>
</tr>
</tbody>
</table>
**THIS LECTURE**

- We saw secure computation protocols
- with tradeoffs
- This time: another tradeoff

<table>
<thead>
<tr>
<th></th>
<th>Rounds</th>
<th>Communication</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDD-based</td>
<td>![Green Thumbs Up]</td>
<td>![Green Thumbs Up]</td>
<td>![Red Thumbs Down]</td>
</tr>
<tr>
<td>Multi-round</td>
<td>![Red Thumbs Down]</td>
<td>![Red Thumbs Down]</td>
<td>![Green Thumbs Up]</td>
</tr>
<tr>
<td>Garbled circuits</td>
<td>![Green Thumbs Up]</td>
<td>![Red Thumbs Down]</td>
<td>![Green Thumbs Up]</td>
</tr>
</tbody>
</table>
RECALL: BOOLEAN CIRCUITS

- A computation model
- Every node evaluates a Boolean function on its inputs
- Output of circuit: top value
- Complexity class of all Boolean functions that can be evaluated by poly-size circuits = class of efficient Boolean functions (P/poly)
Gate implements a function $f: \{0, 1\}^2 \rightarrow \{0, 1\}$

Circuit implements a function $C: \{0, 1\}^n \rightarrow \{0, 1\}$
**BOOLEAN CIRCUITS**

- Fix values of input gates
- Each wire \( w \) obtains recursively a value \( \forall [w] \in \{0, 1\} \)
- Circuit outputs the value of the output wire \( \forall [\text{output}] \)
Fix values of input gates

Each wire \( w \) obtains recursively a value \( \forall [w] \in \{0, 1\} \)

Circuit outputs the value of the output wire \( \forall [\text{output}] \)
◆ Fix values of input gates

◆ Each wire $w$ obtains recursively a value $V[w] \in \{0, 1\}$

◆ Circuit outputs the value of the output wire $V[output]$
 BOOLEAN CIRCUITS

- Fix values of input gates
- Each wire \( w \) obtains recursively a value \( V[w] \in \{0, 1\} \)
- Circuit outputs the value of the output wire \( V[\text{output}] \)
Let $w_i(m)$ be the value of $w_i$, given input assignment $m$

Thus for $m = 010$, 

$C(m) = 1$
 BOOLEAN CIRCUITS

- 16 Boolean functions, so 16 possible gate types

- Possible basis: \{\land, \oplus\}

- \neg x = 1 \oplus x

- \begin{align*}
x \lor y &= \neg(\neg x \land \neg y) = \\
&= 1 \oplus((1 \oplus x) \land (1 \oplus y)) = x \oplus y \oplus (x \land y)
\end{align*}
GARBLED CIRCUITS: GOAL

- Alice knows input $a$
- Bob knows input $b$
- Bob will get to know $C(a, b)$
- ... without learning anything about Alice's input and the intermediate wire values
"Garble" all wire values

- for wire \( w \), 0 \( \rightarrow \) \( K_{w0} \), 1 \( \rightarrow \) \( K_{w1} \)

- except for output wire where \( K_{wj} = j \)

Main idea: Allow Bob to obtain only one \( K_{wj} \) of two garbled values corresponding to each wire
For each Alice's input $w_i$:  
- Alice sends to Bob $K_{w_i a_i}$

For each Bob's input $w_i$:  
- Alice generates and helps Bob to privately obtain $K_{w_i b_i}$

For each internal wire $w$:  
- Alice helps Bob to obtain $K_{w_i b_i}$

$K_{10}, K_{11}, K_{20}, K_{21}, K_{30}, K_{31}, K_{40}, K_{41}, K_{50}, K_{51}, K_{60}, K_{61}$
MORE DETAILS

- For each Alice's input $w_i$:
  - Alice sends to Bob $K_{w_i}a_i$

- For each Bob's input $w_i$:
  - Alice generates and helps Bob to privately obtain $K_{w_i}b_i$

- For each internal wire $w$:
  - Alice helps Bob to obtain $K_{w_ib_i}$
MORE DETAILS

- For each Alice's input $w_i$:
  - Alice sends to Bob $Kw_i a_i$

- For each Bob's input $w_i$:
  - Alice generates and helps Bob to privately obtain $Kw_i b_i$

- For each internal wire $w$:
  - Alice helps Bob to obtain $K_{w_ib_i}$

Protocol 1

Protocol 2
QUIZ: PROTOCOL 1

❖ Let $w$ be one of Bob's input wires
❖ Alice has $K_0, K_1$
❖ Bob has $j$
❖ Quiz: how to transfer $K_j$ to Bob?

Answer: oblivious transfer. CPIR with extra privacy: Bob obtains only one database element and nothing else.
(2, 1)-OT PROTOCOL

This is oblivious transfer in semihonest model. Honest Bob only obtains $M = f_x$
REMARKS: OT

- **Computation**: small number of PK ops
  - As we saw, Paillier is quite expensive, though

- **Communication**: 2 ciphertexts

- **Note**: $|key|$ is short, say 128 bits

- Can use any OT protocol that works with such data
Let $w$ be one of intermediate wires

Alice has keys $K_{u0}, K_{u1}, K_{v0}, K_{v1}, K_{w0}, K_{w1}$

Bob has $K_{ui}, K_{vj}$

Quiz: how to transfer $K_{wk}, k = i \land j$, to Bob?

Answer: garbled gate. Alice sends to Bob $\{\text{Enc}(K_{ui}, \text{Enc}(K_{vj}, K_{wk}))\}$
**GARBLED AND-GATE**

\[ G_{00} \leftarrow \text{AES}(K_{u0}, \text{AES}(K_{v0}, o^{20} \parallel K_{w0})) \]
\[ G_{01} \leftarrow \text{AES}(K_{u0}, \text{AES}(K_{v1}, o^{20} \parallel K_{w0})) \]
\[ G_{10} \leftarrow \text{AES}(K_{u1}, \text{AES}(K_{v0}, o^{20} \parallel K_{w0})) \]
\[ G_{11} \leftarrow \text{AES}(K_{u1}, \text{AES}(K_{v1}, o^{20} \parallel K_{w1})) \]
\[ G[w] \leftarrow \text{random perm. of } G_{ij} \]

**Note 1:** this protocol is non-interactive. Alice can transfer all \( G[w] \) to Bob before Bob knows \( K_{ui}, K_{vj} \).

**Note 2:** Other gate types can be garbled similarly.
REMARKS: GARbled GATE

- **Computation:**
  - **Alice:** 8 AES encryptions
  - **Bob:** 4 AES decryptions in average
  - AES is much \(1000\times\) faster than PK encryption!

- **Communication:** < 500 bits

- Can replace AES with arbitrary faster but secure block cipher
FULL GARBLED CIRCUIT PROTOCOL

Generate new pk for OT
For all Bob's input wires $i$:
1. Prepare OT query $Q(b_i)$
   $c \leftarrow (pk, (Q(b_i))$ for all $i$)

For all wires $w$ of the circuit:
1. Generate random $K_{w_0}, K_{w_1}$
2. Construct $G[w]$ if $w$ is internal
   $d \leftarrow (G[w])$ for all internal wires $w$
   $e \leftarrow (K_{ia_i})$ for all Alice's input wires $i$
For all Bob's input wires $i$:
1. Let $R_i \leftarrow \text{Reply}(Q(b_i), (K_{i0}, K_{i1}))$
   $f \leftarrow (R_i)$ for all Bob's input wires $i$

For all Bob's input wires $i$:
1. $K_{ib_i} \leftarrow \text{Answer}(R_i)$
For all internal wires $w$ of the circuit:
1. $K_{wk} \leftarrow \text{GarbledGate}(K_{ui}, K_{vj}, G[w])$
Return the key of last gate
SECURITY

- **Bob's privacy:**
  - Alice sees only OT queries, so guaranteed by OT security

- **Alice's privacy:**
  - Bob sees AES encryptions and OT replies
  - Security guaranteed by AES security, OT security, and correctness of Alice's operation

- Will omit formal proof of security
EFFICIENCY

 היתר

Round-complexity:

* 2 msg (one msg by Bob, one by Alice) --- optimal # of rounds

Computation:

* |Bob inputs| OT-s, 8*|circuit size| AES-s
* Very good, since AES is fast
  * except when circuit size is really large

Communication:

* < 500 bits per gate, thus large
* Communication is linear in |circuit|, not in |size| :(
**GARBLED CIRCUITS**

- **Rounds:** 2

- **Computation:** $|\text{Bob's input}| \text{ OT, } 8^*|\text{circuit size}| \text{ AES}

- **Communication:** $\Theta (|\text{circuit size}|)$

<table>
<thead>
<tr>
<th></th>
<th>Rounds</th>
<th>Communication</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDD-based</td>
<td>![Thumbs Up]</td>
<td>![Thumbs Up]</td>
<td>![Thumbs Down]</td>
</tr>
<tr>
<td>MPC</td>
<td>![Thumbs Down]</td>
<td>![Thumbs Down]</td>
<td>![Thumbs Up]</td>
</tr>
<tr>
<td>Garbled circuits</td>
<td>![Thumbs Up]</td>
<td>![Thumbs Down]</td>
<td>![Thumbs Up]</td>
</tr>
</tbody>
</table>
EXAMPLE: EQ

- **Alice's input:** $a \in \{0, 1\}^n$
- **Bob's input:** $b \in \{0, 1\}^n$
- **Bob's output:** $[b = a]$
- **Circuit size:** $2n - 1$
- **Cost:** $n$ OT, $8(2n - 1)$ AES
- **Comp. dominated by OT-s**
HOW TO OPTIMIZE?

- **Communication**: linear in \(|\text{circuit size}|\)

- **Task 1**: minimize circuit size

- **Computation** is dominated, depending on circuit size, by OT-s or GarbledGate-s

- **Task 2**: minimize complexity of OT-s

- **Task 3**: minimize complexity of GarbledCircuit's

Non-cryptographic task. Will not elaborate

Cryptographic task. Will elaborate
OT COMPLEXITY MINIMIZATION

- Use more efficient (2, 1)-OT
  - Paillier $\rightarrow$ Elgamal, but can't go much faster

- **OT extension:**
  - we need many OT-s
  - need to reduce the number expensive operations
If input length $n$ is large:

- need to implement $n \gg \kappa$ OT-s

**Question:**

- Can we implement $n$ OT-s by using $n$ "cheap operations" and $\ll n$ OT-s?

Recall: $\kappa$ is the security parameter
RECALL: N OT\textsuperscript{L} PROTOCOLS

For $i = 1$ to $n$:
\[ c_i \leftarrow Q(x_i) \]

For $i = 1$ to $n$:
\[ c_i \leftarrow Q(x_i) \]

\[ (c_1, \ldots, c_n) \]

\[ (f_{i0}, f_{i1}) \in \{0, 1\}^L \]

\[ (R_1, \ldots, R_n) \]

\[ x_i \in \{0, 1\} \]

For $i = 1$ to $n$:
\[ R_i \leftarrow \text{Reply}(c_i, (f_{i0}, f_{i1})) \]

For $i = 1$ to $n$:
\[ R_i \leftarrow \text{Reply}(c_i, (f_{i0}, f_{i1})) \]

\[ M_i \leftarrow \text{Answer}(R_i) \]

For $i = 1$ to $n$:
\[ M_i \leftarrow \text{Answer}(R_i) \]
IDEA: OT EXTENSION

- Alice obtains from Bob, by using $\kappa \ll n$ OT protocols, a number of random bits.
- Alice masks her input by so obtained random bits and her own random input $s$, and sends the masked values to Bob.
- Bob can "unmask" only $\frac{1}{2}$ of the transferred values since the rest depends on $s$. 
For $i = 1$ to $n$:
For $j = 1$ to $\kappa$: $t_{ij} \leftarrow \{0, 1\}$
For $i = 1$ to $\kappa$:
Write $t^i = (t_{i1}, ..., t_{i\kappa})$

Compute $t^i \oplus x$

For $i \in \{1, ..., n\}$:
$(f_{i0}, f_{i1}) \in \{0, 1\}^\kappa$

$$\begin{array}{|c|c|c|}
\hline
\text{t}^1 \oplus x & \text{t}^2 \oplus x & \text{t}^3 \oplus x \\
\hline
\text{t}_{11} & \text{t}_{12} & \text{t}_{13} \\
\text{t}_{21} & \text{t}_{22} & \text{t}_{23} \\
\text{t}_{31} & \text{t}_{32} & \text{t}_{33} \\
\text{t}_{41} & \text{t}_{42} & \text{t}_{43} \\
\text{t}_{51} & \text{t}_{52} & \text{t}_{53} \\
\text{t}_{61} & \text{t}_{62} & \text{t}_{63} \\
\hline
\end{array}$$
N OT^L VIA OT EXTENSION

For \( i = 1 \) to \( n \):
For \( j = 1 \) to \( \kappa \): \( t_{ij} \leftarrow \{0, 1\} \)
For \( i = 1 \) to \( \kappa \):
Write \( t^i = (t_{ii}, ..., t_{ni}) \)
Compute \( t^i \oplus x \)

Alice obtains \( \{A^i = t^i \oplus s_i x\} \) by using \( \kappa \) OT^{n-s}

<table>
<thead>
<tr>
<th>OT(s_1)</th>
<th>OT(s_2)</th>
<th>OT(s_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t^1 \oplus o_x )</td>
<td>( t^1 \oplus i_x )</td>
<td>( t^1 \oplus o_x )</td>
</tr>
<tr>
<td>( t_{11} )</td>
<td>( t_{11} \oplus x_1 )</td>
<td>( t_{12} )</td>
</tr>
<tr>
<td>( t_{21} )</td>
<td>( t_{21} \oplus x_2 )</td>
<td>( t_{22} )</td>
</tr>
<tr>
<td>( t_{31} )</td>
<td>( t_{31} \oplus x_3 )</td>
<td>( t_{32} )</td>
</tr>
<tr>
<td>( t_{41} )</td>
<td>( t_{41} \oplus x_4 )</td>
<td>( t_{42} )</td>
</tr>
<tr>
<td>( t_{51} )</td>
<td>( t_{51} \oplus x_5 )</td>
<td>( t_{52} )</td>
</tr>
<tr>
<td>( t_{61} )</td>
<td>( t_{61} \oplus x_6 )</td>
<td>( t_{62} )</td>
</tr>
</tbody>
</table>
For \( i = 1 \) to \( n \):
For \( j = 1 \) to \( \kappa \): \( t_{ij} \leftarrow \{0, 1\} \)
For \( i = 1 \) to \( \kappa \):
Write \( t^i = (t_{i1}, \ldots, t_{ni}) \)
Compute \( t^i \oplus x \)

Alice obtains \( \{ A^i = t^i \oplus s^i x \} \) by using \( \kappa \) OT\(^n-s\)

Bob knows this if \( x_i = 0 \). Mask \( f_{i0} \) by it

\( A_i = t_i \oplus x_i s \)
\( A_i \oplus s = t_i \oplus (1 \oplus x_i) s \)

Bob knows this if \( x_i = 1 \). Mask \( f_{i1} \) by it
FULL OT EXTENSION

For $i = 1$ to $n$:
  For $j = 1$ to $\kappa$: $t_{ij} \leftarrow \{0, 1\}$
  For $i = 1$ to $\kappa$:
    Write $t^i = (t_{i1}, ..., t_{in})$
    $R_i \leftarrow \text{Reply} \left( q_i, (t^i, t^i \oplus x) \right)$

$s \leftarrow \{0, 1\}^\kappa$

For $i = 1$ to $\kappa$:
  $q_i \leftarrow Q \left( s_i \right)$

$(q_1, ..., q_\kappa)$

$(R_1, ..., R_\kappa)$

$(y_{i0}, y_{i1}, ..., y_{n0}, y_{n1})$

For $i = 1$ to $n$:
  $m_i \leftarrow y_{i,x_i} \oplus H \left( t_i \right)$
Return $m$

$x \in \{0, 1\}^n$

For $i = 1$ to $\kappa$:
  $A^i \leftarrow \text{Answer} \left( R_i \right)$ // = $t^i \oplus s_i \times$
// $A_i = t_i \oplus x_i s$, $A_i \oplus s = t_i \oplus (1 \oplus x_i) s$

For $i = 1$ to $n$:
  $y_{i0} \leftarrow f_{i0} \oplus H \left( A_i \right)$
  $y_{i1} \leftarrow f_{i1} \oplus H \left( A_i \oplus s \right)$

For $i \in \{1, ..., n\}$:
  $(f_{i0}, f_{i1}) \in \{0, 1\}^\mu$
SECURITY

❖ **Alice's privacy:**

❖ Bob sees $q_i$ and $m_i = \ldots + H(\ldots)$

❖ Alice's privacy guaranteed by OT privacy and security of $H$ (*will not elaborate on the latter*)

❖ **Bob's privacy:**

❖ Alice sees $\{t^i \oplus s_i x\}_i$ - no information revealed about $x$ since $t^i$ is random
EFFICIENCY

✧ **Alice:** $\kappa$ OT-s of $n$-bit strings, $2n$ $H$-s

✧ **Bob:** $\kappa$ OT-s of $n$-bit strings, $n$ $H$-s

✧ If $\kappa << n \approx |\text{circuit size}|$ and $\text{Cost}(H) << \text{Cost}(\text{OT})$:
  
  ✧ huge benefit

  ✧ OT dominates the cost of GC

  ✧ OT extension makes GC *many* times faster
Use a more efficient symmetric encryption scheme

gives speedup, but not certain how much

AES is implemented in Intel hardware

Replace AES with a hash function

use Bitcoin hardware
Can we reduce the overhead of GarbledGate?

**Alice**: from 8 AES to smaller number?

**Bob**: from 8 AES\(^{-1}\) to smaller number?
FREEXOR TECHNIQUE

- Alice generates random, secret "global difference" $\Delta$
- She always sets $K_{wI} \leftarrow K_{w0} \oplus \Delta$
- For each XOR gate $w \leftarrow u \oplus v$:
  - set $K_{w0} \leftarrow K_{u0} \oplus K_{v0}$
- The rest of the protocol is unchanged
- Security ok:
  - since Bob never obtains both $K_{w0}, K_{wI}$, then $\Delta$ stays secret
FREEXOR TECHNIQUE

- For each XOR gate: set $K_{w_0} \leftarrow K_{u_0} \oplus K_{v_0}$
  - $K_{u_0} \oplus K_{v_0} = K_{w_0}$
  - $K_{u_1} \oplus K_{v_0} = K_{w_0} \oplus \Delta = K_{w_1}$
  - $K_{u_0} \oplus K_{v_1} = K_{w_0} \oplus \Delta = K_{w_1}$
  - $K_{u_1} \oplus K_{v_1} = K_{w_0} \oplus \Delta \oplus \Delta = K_{w_0}$
- No need to transfer $G[w]$ for XOR gates
- Computation:
  - $\text{AES} \ast O(\text{|non-XOR gates|}) + \text{XOR} \ast O(\text{|XOR gates|})$
WHY RELEVANT?

- `{XOR, AND}` is a basis of all possible gates
  - Boolean versions of `+` and `·`
- **On "average":**
  - Garbled circuits twice more efficient
- **For many circuits, XOR gates dominate strongly**
  - Garbled circuits become many times more efficient
**Idea:** AND is a **symmetric** operation

- $x \land y$ is a function of $x + y$, not of $x$ and $y$ individually
- $(0 \land 0) = 0$, $(0 \land 1) = (1 \land 0) = 0$, $(1 \land 1) = 1$

Thus $x \land y = f_{001}(x + y)$

**FreeSym**

**FreeAdd:** use a random $\Delta$ in all + gates

- $K_{zo}, K_{zi}$
- $f_{001}$
- $K_{wo} = K_{uo} + K_{vo}$, $K_{wi} = K_{wo} + \Delta$, $K_{w2} = K_{wo} + 2\Delta$
- $G_0 \leftarrow \text{AES}(K_{wo}, 0^{20} || K_{zo})$
- $G_1 \leftarrow \text{AES}(K_{wi}, 0^{20} || K_{zo})$
- $G_2 \leftarrow \text{AES}(K_{w2}, 0^{20} || K_{zi})$
- $G[w] \leftarrow \text{random shuffle of } G_i$
- $K_{uo}, K_{ui} = K_{uo} + \Delta$
- $K_{vo}, K_{vi} = K_{vo} + \Delta$
- $8 \text{ AES} \rightarrow 3 \text{ AES}$

**Local computation**

Similar optimization possible with all symmetric gates
## COMPARISON: FREEXOR ETC

<table>
<thead>
<tr>
<th></th>
<th>XOR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>FreeXOR</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>FreeSym</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>&quot;FleXOR&quot; (not covered)</td>
<td>0-2</td>
<td>2</td>
</tr>
</tbody>
</table>
Highly parallel:
- after keys are computed, Alice can compute all garbled gates $G[w]$ in parallel

Many more optimizations than fit on the margins

Very active research area

especially active: efficiency in malicious model

A GPU implementation can process hundreds/thousands of gates in parallel

not covering in this course - just not enough time
GC: OUTRO

- Computationally very efficient

- **Main problem:** huge communication

- **Another problem:** circuit optimization

  - In many cases, it is more natural to work in some other computational model

  - For example: circuit for integer multiplication is large
WHAT NEXT?

- We showed how to compute almost anything securely
  - in the semihonest model

- Starting from the next lecture:
  - What to do when parties are malicious?