Efficient MPC
Optimizations for Garbled Circuits

Claudio Orlandi, Aarhus University
Part 3: Garbled Circuits

• GC: Definitions and Applications

• Garbling gate-by-gate: Basic and optimizations

• Active security 101: simple-cut-and-choose, dual-execution
Garbled Circuit

Cryptographic primitive that allows to evaluate

encrypted functions

on

encrypted inputs
Garbled Circuits

Values in a box are “garbled”

Correct if $z = f(x)$
Application 1: One-time Delegation via GC

Alice

\[ [Z] \leftarrow Ev([F],[X]) \]

Bob(x)

\[ ([F], e, d) \leftarrow Gb(f, r) \]

preprocessing

online

\[ [F] \]

\[ [X] \leftarrow En(e, x) \]

\[ [X] \]

\[ [Z] \]

\[ z = De(d, [Z]) \]
Application 1: One-time Delegation via GC

**Authenticity:**
If A is corrupted and $[Z^*] \leftarrow A([F],[X])$, then $\text{De}([Z^*],d)$ is $f(x)$ or “⊥”
Garbled Circuits: Authenticity

\[ z^* = f(x) \]

OR

\[ z^* = \text{abort} \]
Application 2: Passive Constant Round 2PC (Yao)

Alice(x)

\[ [Z] \leftarrow Ev([F],[X],[Y]) \]
\[ z = De(d,[Z]) \]

2PC (OT)

Bob(y)

\[ ([F],e,d) \leftarrow Gb(f,r) \]
\[ [Y] \leftarrow En(e,y) \]
Application 2: Passive Constant Round 2PC (Yao)

Alice(x)

\[ Z \leftarrow \text{Ev}([F],[X],[Y]) \]
\[ z = \text{De}(d,[Z]) \]

2PC (OT)

Bob(y)

\[ ([F],e,d) \leftarrow \text{Gb}(f,r) \]
\[ [Y] \leftarrow \text{En}(e,y) \]

Bob learns nothing about x!
Application 2: Passive Constant Round 2PC (Yao)

How much information is leaked by GC?
Garbled Circuits: Privacy

\[ \text{Exist Sim s.t. } ([F],[X],d) \sim \text{Sim}(\phi(f),f(x)) \]

\( \phi(f) = f \) or \( \phi(f) = |f| \)
Part 3: Garbled Circuits

• Definitions and Applications

• Garbling gate-by-gate: Basic and optimizations

• Active security 101: simple-cut-and choose, dual-execution
Garbling: Gate-by-gate

\[ z^* \rightarrow [w] \rightarrow w + e \rightarrow [w + e] \rightarrow \ldots \]
PROJECTIVE SCHEMES: CIRCUIT BASED GARBLING/EVALUATIONS
Garbling a Circuit: \(([F], e, d) \leftrightarrow Gb(f)\)

- Choose 2 random keys \(K^i_0, K^i_1\) for each wire in the circuit
  - Input, internal and output wires

- For each gate \(g\) compute
  - \(gg \leftrightarrow Gb(g, L_0, L_1, R_0, R_1, K_0, K_1)\)

- Output
  - \(e=(K^i_0, K^i_1)\) for all input wires
  - \(d=(Z_0, Z_1)\)
  - \([F]=(gg^i)\) for all gates \(i\)
Encoding and Decoding

\[ [X] = \text{En}(e, x) \]
- \( e = \{ K_0^i, K_1^i \} \)
- \( x = \{ x_1, \ldots, x_n \} \)
- \( [X] = \{ K^1_{x_1}, \ldots, K^n_{x_n} \} \)

\[ z = \text{De}(d, [Z]) \]
- \( d = \{ Z_0, Z_1 \} \)
- \( [Z] = \{ K \} \)
- \( z = \)
  - 0 \quad \text{if } K = Z_0, 
  - 1 \quad \text{if } K = Z_1, 
  - "abort" \quad \text{else} \)
Evaluating a GC : \([Z] \leftarrow \text{Ev}([F],[X])\)

- Parse \([X]=\{K_1,\ldots,K_n\}\)
- Parse \([F]=\{gg^i\}\)
- For each gate \(i\) compute
  - \(K \leftarrow \text{Ev}(gg^i, L, R)\)
- Output
  - \(Z\)
INDIVIDUAL GATES GARBLING/EVALUATION
Notation

- A garbled gate is a gadget that given two inputs keys gives you the right output key (and nothing else)

- $gg \leftarrow Gb(g, L_0, L_1, R_0, R_1, Z_0, Z_1)$
- $Z_{g(a,b)} \leftarrow Ev(gg, L_a, R_b)$
- //and not $Z_{1-g(a,b)}$
Yao Gate Garbling (1)

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- AND gate
Yao Gate Garbling (2)

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_0$</td>
<td>$R_0$</td>
<td>$Z_0$</td>
</tr>
<tr>
<td>$L_0$</td>
<td>$R_1$</td>
<td>$Z_0$</td>
</tr>
<tr>
<td>$L_1$</td>
<td>$R_0$</td>
<td>$Z_0$</td>
</tr>
<tr>
<td>$L_1$</td>
<td>$R_1$</td>
<td>$Z_1$</td>
</tr>
</tbody>
</table>

• Choose labels (e.g., 128 bits strings) for every value on every wire
Yao Gate Garbling (3)

\[
\begin{array}{|c|}
\hline
C \\
C_1 = G(L_0,R_0) \oplus Z_0 \\
C_2 = G(L_0,R_1) \oplus Z_0 \\
C_3 = G(L_1,R_0) \oplus Z_0 \\
C_4 = G(L_1,R_1) \oplus Z_1 \\
\hline
\end{array}
\]

- Encrypt the output key with the input keys
  - G is some "key derivation function" so that the encryption is secure
Yao Gate Garbling (4)

\[
\begin{align*}
C_1 &= G(L_0, R_0) \oplus (Z_0, 0^k) \\
C_2 &= G(L_0, R_1) \oplus (Z_0, 0^k) \\
C_3 &= G(L_1, R_0) \oplus (Z_0, 0^k) \\
C_4 &= G(L_1, R_1) \oplus (Z_1, 0^k)
\end{align*}
\]

- Add redundancy (later used to check if decryption is successful)
Yao Gate Garbling (5)

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 = G(L_0, R_0) \oplus (Z_0, 0^k)$</td>
</tr>
<tr>
<td>$C_2 = G(L_0, R_1) \oplus (Z_0, 0^k)$</td>
</tr>
<tr>
<td>$C_3 = G(L_1, R_0) \oplus (Z_0, 0^k)$</td>
</tr>
<tr>
<td>$C_4 = G(L_1, R_1) \oplus (Z_1, 0^k)$</td>
</tr>
</tbody>
</table>

$C'_1, C'_2, C'_3, C'_4 = \text{perm}(C_1, C_2, C_3, C_4)$

- Permute the order of the ciphertexts (to hide information about inputs/outputs)
Yao Gate Evaluation (1)

Eval\((gg, L_a, R_b) //not a,b\)

- For i=1..4
  - \((K,t)=C'_i \oplus G(L_a,R_b)\)
  - If \(t=0^k\) output \(K\)

- Output is correct:
  - \(t=0^k\) only for right row

- Evaluator learns nothing else:
  - Encryption + permutation

<table>
<thead>
<tr>
<th>gg (permuted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_1 = G(L_0,R_0) \oplus (K_1,0^k))</td>
</tr>
<tr>
<td>(C_2 = G(L_0,R_1) \oplus (K_1,0^k))</td>
</tr>
<tr>
<td>(C_3 = G(L_1,R_0) \oplus (K_1,0^k))</td>
</tr>
<tr>
<td>(C_4 = G(L_1,R_1) \oplus (K_0,0^k))</td>
</tr>
</tbody>
</table>
### Efficiency

|       | $|g|g|$ | G/Gb | G/Eval | Assumption on G |
|-------|-------|------|--------|------------------|
| Classic | 8k    | 4    | 4      | Standard         |
Point-and-permute

- **Problem**: Evaluator needs to try to decrypt all 4 rows
- **Solution**: add permutation bits to keys

\[

gg \leftarrow \text{Gb}(g, L_0, L_1, p, R_0, R_1, q, Z_0, Z_1, r)
\]

\[
(Z_{g(a,b)}, r \oplus ab) \leftarrow \text{Ev}(gg, L_a, a \oplus p, R_b, b \oplus q)
\]
## Point-and-permute Garbling (4)

<table>
<thead>
<tr>
<th>( C )</th>
<th>( C_1 = G(L_0, R_0) \oplus (Z_0, r \oplus 0) )</th>
<th>( C_2 = G(L_0, R_1) \oplus (Z_0, r \oplus 0) )</th>
<th>( C_3 = G(L_1, R_0) \oplus (Z_0, r \oplus 0) )</th>
<th>( C_4 = G(L_1, R_1) \oplus (Z_1, r \oplus 1) )</th>
</tr>
</thead>
</table>

- Remove redundancy
- Add random permutation bit
Point-and-permute Garbling (5)

<table>
<thead>
<tr>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C’<em>0 = G(L_p, R_q) \oplus (Z</em>{p \cdot q}, r \oplus p \cdot q) )</td>
</tr>
<tr>
<td>( C’<em>1 = G(L_p, R</em>{!q}) \oplus (Z_{p \cdot !q}, r \oplus p \cdot !q) )</td>
</tr>
<tr>
<td>( C’<em>2 = G(L</em>{!p}, R_q) \oplus (Z_{!p \cdot q}, r \oplus !p \cdot q) )</td>
</tr>
<tr>
<td>( C’<em>3 = G(L</em>{!p}, R_{!q}) \oplus (Z_{p \cdot !q}, r \oplus !p \cdot !q) )</td>
</tr>
</tbody>
</table>

- Permute rows using \( p, q \)
Point-and-permute Evaluation

Eval(gg, L, u, R, v) //not a,b

• (Z,r)=C’₂u+v ⊕ G(L,R)

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>C’₀ = G(Lₚ,Rₚ) ⊕ (Zₚ,p, r ⊕ p·q )</td>
</tr>
<tr>
<td>C’₁ = G(Lₚ,R!q) ⊕ (Zₚ!q, r ⊕ p·!q )</td>
</tr>
<tr>
<td>C’₂ = G(L!ₚ,Rₚ) ⊕ (Z!ₚ,q, r ⊕ !p·q )</td>
</tr>
<tr>
<td>C’₃ = G(L!ₚ,R!q) ⊕ (Z!ₚ!q, r ⊕ !p·!q )</td>
</tr>
</tbody>
</table>

• Output is correct:
  – (Check permutation)

• Privacy:
  – u=p⊕a, v=q⊕b
  – p,q are “one time pads” for a,b
# Efficiency

<table>
<thead>
<tr>
<th>l</th>
<th>gg</th>
<th></th>
<th>G/Gb</th>
<th>G/Eval</th>
<th>Assumption on G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic</td>
<td>8k</td>
<td>4</td>
<td>4</td>
<td>Standard</td>
<td></td>
</tr>
<tr>
<td>P&amp;P</td>
<td>4k</td>
<td>4</td>
<td>1</td>
<td>Standard</td>
<td></td>
</tr>
</tbody>
</table>
GARBLING OPTIMIZATIONS:
SIMPLE GARBLED ROW REDUCTION
Changing the syntax

- **Problem**: each gg is 4 ciphertexts
- **Solution**: define output key pseudorandomly as functions of input keys, reduce comm. complexity

\[(gg, Z_0, Z_1) \leftarrow Gb(g, L_0, L_1, R_0, R_1)\]
\[(Z_{g(a,b)}) \leftarrow Ev(gg, L_a, R_b)\]

Note, now garbling cannot be done in parallel anymore!
Garbling a Circuit: \([F], e, d \leftarrow Gb(f)\)

- Choose 2 random keys \(K^i_0, K^i_1\) for each wire in the circuit
  - *Input wire only!*

- For each gate \(g\) compute
  - \((gg, K_0, K_1) \leftarrow Gb(g, L_0, L_1, R_0, R_1)\)

- Output
  - \(e=(K^i_0, K^i_1)\) for all input wires
  - \(d=(Z_0, Z_1)\)
  - \([F]=(gg^i)\) for all gates \(i\)
Yao Gate Garbling (3)

\[
\begin{array}{|c|}
\hline
C \\
\hline
C_1 = G(L_0, R_0) \oplus Z_0 \\
C_2 = G(L_0, R_1) \oplus Z_0 \\
C_3 = G(L_1, R_0) \oplus Z_0 \\
C_4 = G(L_1, R_1) \oplus Z_1 \\
\hline
\end{array}
\]

• Encrypt the output key with the input keys
Garbled Row Reduction Garbling

- Define output keys as function of input keys
  - (compatible with p&p)
  - Can reduce 2 rows, but 1 is compatible with Free-XOR (coming up!)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>$Z_0 = G(L_0, R_0)$ ($C_1=0^k$)</td>
<td></td>
</tr>
<tr>
<td>$C_2 = G(L_0, R_1) \oplus Z_0$</td>
<td></td>
</tr>
<tr>
<td>$C_3 = G(L_1, R_0) \oplus Z_0$</td>
<td></td>
</tr>
<tr>
<td>$C_4 = G(L_1, R_1) \oplus Z_1$</td>
<td></td>
</tr>
</tbody>
</table>
## Efficiency

|       | $|g_g|$ | G/Gb | G/Eval | Assumption on G |
|-------|------|------|--------|-----------------|
| Classic | 8k   | 4    | 4      | Standard        |
| P&P    | 4k   | 4    | 1      | Standard        |
| +GRR   | 3k/2k| 4    | 1      | Standard        |
GARBLING OPTIMIZATIONS:
FREE XOR
Free-XOR

• **Problem:** with secret sharing linear gates are for free. What about GC?

• **Solution:** introduce correlation between keys, make XOR computation “free”

\[
L_0 = L_0 \oplus \Delta \\
L_1 = L_0 \oplus \Delta \\
R_0 = R_0 \oplus \Delta \\
R_1 = R_0 \oplus \Delta
\]

\[
(gg, Z_0) \leftarrow Gb(g, L_0, R_0, \Delta) \\
(Z_{g(a,b)}) \leftarrow Ev(gg, L_a, R_b)
\]
Changing syntax, again!

- Choose 1 random key $K^i_0$ for each input wire in the circuit
  - And global difference $\Delta$

- For each gate $g$ compute
  - $(gg,K_0) \leftarrow Gb(g,L_0,R_0,\Delta)$

- Output
  - $e=(K^i_0,K^i_1)$ for all input wires
  - $d=(Z_0,Z_1)$
  - $[F]=(gg^i)$ for all gates $i$
What about AND Gates?

- Like before, but requires “circular security assumption”

- Evaluator sees

  \[ L_0, R_0, Z_0, \text{ and } G(L_0 \oplus \Delta, R_0 \oplus \Delta) \oplus Z_0 \oplus \Delta \]

- And should not be able to compute \( \Delta \)!
- Effectively an encryption of \( \Delta \) under \( \Delta \)!
Garbling/Evaluating XOR Gates

\[ L_0 \]
\[ L_1 = L_0 \oplus \Delta \]
\[ R_0 \]
\[ R_1 = R_0 \oplus \Delta \]

**Gb(XOR, L_0, R_0, \Delta)**
- Output \( Z_0 = L_0 \oplus R_0 \)
- (gg is empty)

**Ev(XOR, L_a, R_b, \Delta)**
- Output \( Z_{a \oplus b} = L_a \oplus R_b \)

\[ L_a \oplus R_b = L_0 \oplus a\Delta \oplus R_0 \oplus b\Delta = Z_0 \oplus (a \oplus b)\Delta = Z_{a \oplus b} \]
# Efficiency

<table>
<thead>
<tr>
<th></th>
<th>AND</th>
<th>XOR</th>
<th>Assumption on G</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>gg</td>
<td>**</td>
<td>G/Gb</td>
</tr>
<tr>
<td>Classic</td>
<td>8k</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>P&amp;P</td>
<td>4k</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>+GRR</td>
<td>3k/2k</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>+Free-XOR</td>
<td>3k</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Privacy Free Garbing

• In some application (example, the delegation of computation from the first slide) we don’t care about hiding the input/output of the circuit to the evaluator.

• Can we construct more efficient garbling if we don’t care about *privacy*, but only *authenticity*?
Privacy Free with Free XOR

• For XOR-gates it is hard to do better than *free*-XOR

• What about AND gates?

• Let \( c = \text{AND}(a, b) \) then
  - If \( a = 0 \) \( \rightarrow \) \( c = 0 \)
  - If \( a = 1 \) \( \rightarrow \) \( c = b \)
Privacy-Free AND gates

\( L_0 \)
\( L_1 = L_0 \oplus \Delta \)
\( R_0 \)
\( R_1 = R_0 \oplus \Delta \)

\( \text{AND} \)

\( Z_0 \)
\( Z_1 = Z_0 \oplus \Delta \)

\( (gg, Z_0) \leftarrow Gb(g, L_0, R_0, \Delta) \)
\( Z_{ab} \leftarrow Ev(gg, L_a, a, R_b, b) \)

\( Gb(\text{AND}, L_0, R_0, \Delta) \)
- \( Z_0 = G(L_0) \quad // \ GRR \)
- \( C = G(L_1) \oplus Z_0 \oplus R_0 \)

\( Ev(gg, L_a, a, R_b, b, \Delta) \)
- If \( a = 0 \) : output
  \( Z_0 = G(L_0) \)
- If \( a = 1 \) : output
  \( Z_b = C \oplus G(L_1) \oplus R_b \)

\( Z_b = C \oplus G(L_1) \oplus R_b = Z_0 \oplus R_0 \oplus (R_0 \oplus b \Delta) = Z_b \)
## Efficiency

<table>
<thead>
<tr>
<th></th>
<th>AND</th>
<th></th>
<th>XOR</th>
<th></th>
<th>Assumption on G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>g</td>
<td>g</td>
<td>$</td>
<td>G/Gb</td>
</tr>
<tr>
<td>Classic</td>
<td>8k</td>
<td>4</td>
<td>4</td>
<td>8k</td>
<td>4</td>
</tr>
<tr>
<td>P&amp;P</td>
<td>4k</td>
<td>4</td>
<td>1</td>
<td>4k</td>
<td>4</td>
</tr>
<tr>
<td>+GRR</td>
<td>3k/2k</td>
<td>4</td>
<td>1</td>
<td>3k/2</td>
<td>4</td>
</tr>
<tr>
<td>+Free-XOR</td>
<td>3k</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Privacy-Free*</td>
<td>k</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Privacy-Free Garbling: Extension?

• Can the same trick help us also in garbling for 2PC?

• Can we let the evaluator learn the bit of some internal wires?
  – No!

• Can we let the evaluator learn a one-time pad encryption of some internal wire?
  – Sure, why not!
Half-Gate (Two Halves Make a Whole)

• Note that we can write:
  \[ a \cdot b = (a \cdot r) \oplus (a \cdot (r \oplus b)) \]

  - \( r \) known to garbler
    \( \implies \text{How to garble efficiently?} \)
  - \( r \oplus b \) known to evaluator \( \implies \text{can use PF garbling} \)

• 1 AND \( \rightarrow \) 2 ANDs. How is this better?
  – The garbled can choose a random \( r \) at garbling time
  – Make sure that the evaluator learns \( (r \oplus b) \)
  – How? Use the permutation bits from point&permute!
How to garble with hidden constant

• The garbled knows r and wants to garble $c = a \cdot r$
  – If $r = 0 \rightarrow c=0$
  – If $r = 1 \rightarrow c=a$

• How to garble an unary gate, which is either the 0 gate or the identity gate depending on r?
Garbling AND with hidden-constant

Let $p=0$ for simplicity

$$C_a \oplus G(L_a) = Z_0 \oplus a(r\Delta) = Z_{ar}$$
Efficiency

<table>
<thead>
<tr>
<th>Assumption on G</th>
<th>AND</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(</td>
<td>gg</td>
</tr>
<tr>
<td>Classic</td>
<td>8k</td>
<td>4</td>
</tr>
<tr>
<td>P&amp;P</td>
<td>4k</td>
<td>4</td>
</tr>
<tr>
<td>+GRR</td>
<td>3k/2k</td>
<td>4</td>
</tr>
<tr>
<td>+Free-XOR</td>
<td>3k</td>
<td>4</td>
</tr>
<tr>
<td>Privacy-Free*</td>
<td>k</td>
<td>4</td>
</tr>
<tr>
<td>Half-Gate</td>
<td>2k</td>
<td>4</td>
</tr>
<tr>
<td>GLP</td>
<td>2k</td>
<td>4</td>
</tr>
</tbody>
</table>

- GLP : Fast Garbling of Circuits Under Standard Assumptions (Gueron, Lindell, Pinkas)
- (The measure of G in this table is somehow arbitrary, in practice the size of the input to G makes a difference in runtime)
Part 3: Garbled Circuits

• Definitions and Applications

• Garbling gate-by-gate: Basic and optimizations

• Active security 101: simple-cut-and choose, dual-execution
ACTIVE ATTACKS VS YAO
Yao´s protocol

Alice

\[ [Z] \leftarrow \text{Ev}([F],[X],[Y]) \]
\[ z = \text{De}(d,[Z]) \]

Bob

\[ ([F],e,d) \leftarrow \text{Gb}(f,r) \]
\[ [Y] \leftarrow \text{En}(e,y) \]

Passive Security
Only 1 GC!
Constant round!
Very fast!
Active security of Yao

Cannot really cheat!
Active security of Yao (v2, Bob gets output)

Still can’t cheat, authenticity!
Garbled Circuits: Authenticy

For all corrupt Ev, 
\[ z^* = f(x) \text{ or } z^* = \text{abort} \]
Active security of Yao

What if B is corrupted?
Insecurity 1 (wrong f)

Alice(x)

$\leftarrow Z \leftarrow Ev([G],[X],[Y])$

$z = De(d,[Z])$

Bob(y)

$([G],e,d) \leftarrow Gb( g,r )$

$[Y] \leftarrow En(e,y)$

$g \neq f$

$z \neq f(x,y)$
Insecurity 2 (selective failure)

\[ (\text{Alice}(x) \leftrightarrow \text{Bob}(y)) \]

\[ [Z] \leftrightarrow \text{Ev}([G],[X],[Y]) \]
\[ z = \text{De}(d,[Z]) \]

\[ [X] = K_x \]

\[ K_0, K_1 \]

\[ (\text{F},e,d) \leftrightarrow \text{Gb}(f,r) \]
\[ [Y] \leftrightarrow \text{En}(e,y) \]

\[ [G], [Y], d \]
Insecurity 2 (selective failure)

Alice(x)

\[ [Z^*] \leftarrow Ev([G],[X^*],[Y]) \]

\[ z^* = De(d,[Z^*]) \]

Bob(y)

\([F], e, d) \leftrightarrow Gb(f,r) \]

\([Y] \leftrightarrow En(e,y) \)

\[ (K_0,K^*) \]

\[ x \]

\[ [X^*] \]

\[ [G], [Y], d \]

x=0 \(\rightarrow\) \( z^* = f(x,y) \)

x=1 \(\rightarrow\) \( z^* = \text{abort} \)
SIMPLE TRICKS FOR ACTIVE SECURITY
ZKGC (Alice proves $f(x) = z$)

Alice($x$) 

$[Z] \leftarrow \text{Ev}([F],[X])$

OT

Bob($\ )$

$([F],e,d) \leftarrow \text{Gb}(f,r)$

$z = \text{De}(d,[Z])$

Bob has no input!
ZKGC (Alice proves $f(x)=z$)

Alice(?)

Bob( )
$([F], e, d) \leftarrow Gb( f, r )$

$z^* = De(d, [Z^*])$

OT

$X$

$e$

$[X]$}

$[F]$

$[Z^*]$

Authenticity!
ZKGC (Alice proves $f(x)=z$)

Corrupt B can change $f$ with $g$. Break privacy!
ZKGC (Alice proves $f(x)=z$)

**Commitment**

- Alice
  - $[Z] \leftarrow \text{Ev}([F],[X])$
  - If $[F]!=$Gb($f,r$) abort
  - else

**OT**

- $[F]$ and $[Z]$

**Opening**

- Bob
  - $([F], e, d) \leftarrow \text{Gb}( f, r )$
  - $z=\text{De}(d,[Z])$

**Active security! Only 1 GC!**
Cut-And-Choose
2PC, simple cut-and-choose

If $G_b(f, r_j) \neq [F]_j$
abort
else
$[Z]_j \leftarrow Ev([F]_j, [X]_j, [Y]_j)$
$z = De(d_j, [Z]_j)$

$( [F]_i, e_i, d_i ) \leftarrow G_b( f, r_i )$

$(X_1), (X_2)$

$x \leftarrow OT$
$e_1, e_2$

$[F]_1, [F]_2, d_1, d_2$

$rand j$

$r_{-j}, [Y]_j$

$[Y]_j \leftarrow En(e_j, y)$
2PC, simple cut-and-choose

If $G_b(f, r_j) \neq [F]_j$ abort
else
$[Z]_j \leftarrow \text{Ev}([F]_j, [X]_j, [Y]_j)$

$z = \text{De}(d_j, [Z]_j)$

Corrupt Bob only wins with probability $1/2$
2PC, cut-and-choose

• Simple cut-and-choose
  – Garble $k$, check $k-1$, evaluate 1.
  – Security $1-1/k$

• Advanced cut-and-choose (see references)
  – Garble $2k$, check $k$, evaluate $k$
  – Output majority result
  – Security with $2^{-O(k)}$
  – (Need mechanisms to ensure the same input is used!)
Dual Execution
Alice

\[[Z_1] \leftarrow Ev([F_1],[X_1],[Y_1])\]

\[(F_2,e_2,d_2) \leftarrow Gb(f,r_2)\]

\[[X_2] \leftarrow En(e_2,x)\]

\[[Z_1],d_2 \leftarrow De(d_1,[Z_1])\]

\[z/abort \leftarrow De(d_2,[Z_2])\]

Bob

\(([F_1],e_1,d_1) \leftarrow Gb(f,r_1)\]

\[[Y_1] \leftarrow En(e_1,y)\]

\[[Z_2] \leftarrow Ev([F_2],[X_2],[Y_2])\]

\[z/abort \leftarrow De(d_2,[Z_2])\]
Alice

Bob

\( ([F_1], e_1, d_1) \leftarrow Gb(f, r_1) \)

\([Y_1] \leftarrow En(e_1, y) \)

\( X_1 \)

\( e_1 \)

\( [X_1] \)

\( [F_1], [Y_1] \)

\( x \)

\( e_2 \)

\( y \)

\( [Y_2] \)

\( e_2 \)

\( [F^*, X_2] \)

\( [Y_2] \)

\( [Z^*] \leftarrow Ev([F_2], [X_2], [Y_2]) \)

\( [Z^*] = [Z] \iff y = 0 \)

1 bit leakage

Selective failure

\( [Z^*] = [Z] \iff y = 0 \)

\( z/abort \)

\( z/abort \)

\( De(d_1, [Z_1]) \)

\( == \)

\( De(d_2, [Z_2]) \)
Forge And Lose
2PC, forge-and-lose (idea)

Alice

\{ [F]_i \}_i

\[ Z_i \leftarrow Ev([F]_i, [X]_i, [Y]_i) \]

Bob

\(([F]_i, e_i, d_i) \leftarrow Gb(f, r_i)\)

\[ [Y]_i \leftarrow En(e_i, y) \]

SMAC

"Punish Bob"
If exist i, j
\[ De(d_i, [Z]_i) \]
\[ \neq \]
\[ De(d_j, [Z]_j) \]
2PC, forge-and-lose (idea)

**Alice**

\[
\{ [F]_i \}_i
\]

\[
[Z]_i \leftarrow Ev([F]_i, [X]_i, [Y]_i)
\]

**Bob**

\[
([F]_i, e_i, d_i) \leftarrow Gb(f, r_i)
\]

\[
[Y]_i \leftarrow En(e_i, y)
\]

Cut-and-Choose
Generate k, check k/2

SMAC

"Punish Bob"
If exist i, j
\[
De(d_i, [Z]_i)
\]
\[
\ne (d_j, [Z]_j)
\]

Whp at least one unchecked circuit is correct
→ real output will be used in SMAC
2PC, forge-and-lose (idea)

Alice

Bob

\((\lfloor F \rfloor_i, e_i, d_i) \leftarrow Gb(f, r_i)\)

\([Y]_i \leftarrow \text{En}(e_i, y)\)

SMAC

"Punish Bob"

If exist \(i, j\)

\(\text{De}(d_i, [Z]_i) \neq \text{De}(d_j, [Z]_j)\)

Authenticity ⇒ Corrupt Alice cannot blame honest Bob

Cut-and-Choose

Generate \(k\), check \(k/2\)
Summary

• Garbling Schemes
  – Definitions and Applications

• Efficient Garbling Techniques
  – Point&Permute, Garbled Row Reduction, Free-XOR, Half-Gate, Privacy-Free, ...

• How to use Garbled Circuits with active corruptions
Primary References

- Cryptographic Computing, lecture notes, http://orlandi.dk/crycom (with theory and programming exercises)
- A Brief History of Practical Garbled Circuit Optimizations (Rosulek)
- Fast Cut-and-Choose-Based Protocols for Malicious and Covert Adversaries (Lindell)
- Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates (Zahur et al.)
- Privacy-Free Garbled Circuits with Applications to Efficient Zero-Knowledge (Frederiksen et al.)
- Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently (Jawurek et al.)
- Improved Garbled Circuit: Free XOR Gates and Applications (Kolesnikov et al.)
- Foundations of Garbled Circuits (Bellare et al.)
Other References

• Fast Garbling of Circuits Under Standard Assumptions (Gueron et al.)
• Garbling Gadgets for Boolean and Arithmetic Circuits (Ball et al.)
• FleXOR: Flexible Garbling for XOR Gates That Beats Free-XOR (Kolesnikov et al.)
• MiniLEGO: Efficient Secure Two-Party Computation From General Assumptions (Frederiksen et al.)
• Secure Two-Party Computation with Reusable Bit-Commitments, via a Cut-and-Choose with Forge-and-Lose Technique (Brandao)
• Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer (Lindell, Pinkas)
• An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries (Lindell, Pinkas)
• Efficiency Tradeoffs for Malicious Two-Party Computation (Mohassel et al.)