Efficient Secure Two-Party Computation

Claudio Orlandi, Aarhus University
Plan for the next 3 hours...

• **Part 1: Secure Computation with a Trusted Dealer**
  – Warmup: One-Time Truth Tables
  – Evaluating Circuits with Beaver’s trick
  – MAC-then-Compute for Active Security

• **Part 2: Oblivious Transfer**
  – OT: Definitions and Applications
  – Passive Secure OT Extension
  – OT Protocols from DDH (Naor-Pinkas/PVW)

• **Part 3: Garbled Circuits**
  – GC: Definitions and Applications
  – Garbling gate-by-gate: Basic and optimizations
  – Active security 101: simple-cut-and choose, dual-execution
Want more?

• Cryptographic Computing – Foundations
  – http://orlandi.dk/crycom
  – Programming & Theory Exercises
  – Will be happy to answer questions by mail!
Online Poker

2♠, 5♠, 2♥, 5♥, J♦

Q♠, Q♣, 7♠, 3♥, 2♦

10♠, 9♣, 8♦, 7♦, 6♦

3♠, 4♠, 7♥, Q♦, 10♦
Poker with Pirates

2♠, 5♠, 2♥, 5♥, J♦
Q♠, Q♣, 7♠, 3♥, 2♦,
10♠, 9♣, 8♠, 7♦, 6♦
A♠, A♣, A♥, A♦, K♦
Secure Computation

2♠, 5♠, 2♥, 5♥, J♠

Q♠, Q♣, 7♠, 3♥, 2♦,

10♠, 9♣, 8♥, 7♦, 6♦

3♠, 4♣, 7♥, Q♦, 10♠
Problem: Sick people forget to claim their insurance money.

Solution: Insurances and hospitals could periodically compare their data to find and help these people.

Privacy Issue: insurance and medical records are sensitive data! No other information than what is strictly necessary must be disclosed!
MPC Goes Live (2008)

Bogetoft et al.
“Multiparty Computation Goes Live”

• January 2008
• **Problem**: determine market price of sugar beets contracts
• 1200 farmers
• Computation: 30 minutes
• **Weak security 😞**
  – Passive adversary
  – Honest majority
Sharemind

• **Benchmarking**
  – ICT Companies,
  – Public sector

• Satellite collisions

• ...
MPC in PRACTICE

- **Partisia**: Secure auctions
- **Dyadic Security**: Server breach mitigation
- **Sharemind**: Benchmarking, satellite collision
- **SAP**: Private smart-metering
- **IBM**: Secure cloud computing
- **Google?**: (looking forward to rwc2017)
Secure Computation

- Privacy
- Correctness
- Input independence
- ...

\[ f(x, y) \]

\[ 8dx2rru3d0fW2TS \]
\[ muv6tbWg32flqlo \]
\[ s1e4xq13OtTzoJc \]
What kind of Secure Computation?

• **Dishonest majority**
  – The adversary can corrupt up to n-1 participants (n=2).

• **Static Corruptions**
  – The adversary chooses which party is corrupted before the protocol starts.

• **Passive & Active Corruptions**
  – Adversary follows the protocol vs. 
    (aka semi-honest, honest-but-curious)
  – Adversary can behave arbitrarily 
    (aka malicious, byzantine)

• **No guarantees of fairness or termination**
  – Security with abort
Trusted Party

\[ f(x, y) \]

Trusted Dealer

\((r_A, r_B) \leftarrow D\)
Online Phase

Preprocessing

- Independent of \( x,y \)
- Typically only depends on size of \( f \)
- Uses public key crypto technology \((slower)\)

\[
\begin{align*}
  r_A & \quad r_B \\
  f(x, y) & \quad x, y
\end{align*}
\]

- Uses only information theoretic tools \((order of magn. faster)\)
Part 1: Secure Computation with a Trusted Dealer

• **Warmup:** One-Time Truth Tables

• Evaluating Circuits with Beaver’s trick

• MAC-then-Compute for Active Security
“The simplest 2PC protocol ever”

\[(r_A, r_B) \leftarrow D\]

\[r_A \quad r_B\]

\[x \quad y\]

\[f(x, y)\]
“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

1) Write the truth table of the function $F$ you want to compute

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
“The simplest 2PC protocol ever” OTTT
(Preprocessing phase)

2) Pick random \((r, s)\), rotate rows and columns

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

\(s=3\)

\(r=1\)
"The simplest 2PC protocol ever" OTTT (Preprocessing phase)

3) Secret share the truth table i.e.,

Pick \( T_1 \) at random, and let

\[
\begin{array}{cccc}
1 & 4 & 4 & 1 \\
2 & 2 & 2 & 3 \\
0 & 0 & 4 & 3 \\
0 & 0 & 4 & 1 \\
\end{array}
\]

\[ T_2 = \begin{array}{c}
T_1
\end{array} \]

\[ - \]
"The simplest 2PC protocol ever" (Online phase)

\[ u = x + r \]
\[ v = y + s \]

"Privacy": inputs masked w/uniform random values

\[ T1 \]
\[ T2 \]

Correctness: by construction

output \( f(x,y) = T1[u,v] + T2[u,v] \)
What about active security?

\[
\begin{align*}
T_1, r \\
u &= x + r \\
v &= y + s + e_1 \\
T_2[u,v] + e_2 \\
T_2, s
\end{align*}
\]
Is this cheating?

- \( v = y + s + e_1 = (y+e_1) + s = y' + s \)
  - Input substitution, not really cheating a
    (see formal definition)

- \( M2[u,v] + e_2 \)
  - Changes output to \( z' = f(x,y) + e_2 \)
  - Example: \( f(x,y)=1 \) iff \( x=y \) (e.g. pwd check)
  - \( e_2=1 \) the output is 1 whp (login without pwd!)
    * Clearly breach of security!
Force Bob to send the right value

- **Problem**: Bob can send the wrong shares
- **Solution**: use MACs
  - e.g. \( m = ax + b \) with \( (a,b) \leftarrow F \)

Abort if \( m' \neq ax' + b \)
If \( (M[u,v] = A[u,v] \times T2[u,v] + B[u,v]) \)

output \( f(x,y) = T1[u,v] + T2[u,v] \)

else

abort

Statistical security vs. malicious Bob w.p. \( 1-1/|F| \)
“The simplest 2PC protocol ever” OTTT

- Optimal communication complexity 😊

- Storage exponential in input size 😞

→ Represent function using circuit instead of truth table!
Part 1: Secure Computation with a Trusted Dealer

- Warmup: One-Time Truth Tables
- Evaluating Circuits with Beaver’s trick
- MAC-then-Compute for Active Security
Circuit based computation
Invariant

• For each *wire* \( x \) in the circuit we have
  
  – \([x] : = (x_A, x_B)\) // read “\( x \) in a box”
  
  – Where Alice holds \( x_A \)
  
  – Bob holds \( x_B \)
  
  – Such that \( x_A + x_B = x \)

• Notation overload:
  
  – \( x \) is both the r-value and the l-value of \( x \)
  
  – use \( n(x) \) for name of \( x \) and \( v(x) \) for value of \( x \) when in doubt.
  
  – Then \([n(x)] = (x_A, x_B)\) such that \( x_A + x_B = v(x) \)
Circuit Evaluation
(Online phase)

1)  \([x] \leftarrow \text{Input}(A,x)\):
   
   – chooses random \(x_B\) and send it to Bob
   – set \(x_A = x + x_B\)  \(\text{// symmetric for Bob}\)

   Alice only sends a random bit! “Clearly” secure

2)  \(z \leftarrow \text{Open}(A,[z])\):
   
   – Bob sends \(z_B\)
   – Alice outputs \(z = z_A + z_B\)  \(\text{// symmetric for Bob}\)

   Alice should learn \(z\) anyway! “Clearly” secure
Circuit Evaluation
(Online phase)

2) \( [z] \leftarrow \text{Add}([x],[y]) \)  
   \[ z = x + y \]  
   // at the end \( z = x + y \)

   - Alice computes \( z_A = x_A + y_A \)
   - Bob computes \( z_B = x_B + y_B \)

   - We write \( [z] = [x] + [y] \)

No interaction! “Clearly” secure

“for free” : only a local addition!
Circuit Evaluation
(Online phase)

2a) $[z] \leftarrow \text{Mul}(a,[x])$  \hspace{1cm} // at the end $z = a \times x$

- Alice computes $z_A = a \times x_A$
- Bob computes $z_B = a \times x_B$

2c) $[z] \leftarrow \text{Add}(a,[x])$  \hspace{1cm} // at the end $z = a + x$

- Alice computes $z_A = a + x_A$
- Bob computes $z_B = x_B$
3) Multiplication?

How to compute $[z]=[xy]$?

Alice, Bob should compute

$$z_A + z_B = (x_A + x_B)(y_A + y_B)$$

$$= x_A y_A + x_B y_A + x_A y_B + x_B y_B$$

Alice can compute this

Bob can compute this

How do we compute this?
3) $[z] \leftarrow \text{Mul}([x],[y])$:

1. Get $[a],[b],[c]$ with $c=ab$ from trusted dealer

2. $e = \text{Open}([a]+[x])$

3. $d = \text{Open}([b]+[y])$

4. Compute $[z] = [c] + e[y] + d[x] - ed$

   $ab + (ay+xy) + (bx+xy) - (ab+ay+bx+xy)$
Part 1: Secure Computation with a Trusted Dealer

• Warmup: One-Time Truth Tables

• Evaluating Circuits with Beaver’s trick

• MAC-then-Compute for Active Security
Secure Computation

\[
\begin{align*}
&x_1 y_1 + e \\
&x_2 y_2 \\
&x_3 y_3 \\
&x_4 y_4 \\
&x_5 y_5 \\
&z^* = w + e
\end{align*}
\]
Active Security?

- “Privacy?”
  - even a malicious Bob does not learn anything 😊

- “Correctness?”
  - a corrupted Bob can change his share during any “Open” (both final result or during multiplication) leading the final output to be incorrect 😞
Problem

2) $z \leftarrow \text{Open}(A,[z])$:

- Bob sends $z_B + e$
- Alice outputs $z = z_A + z_B + e$  // error change output distribution in way that cannot be simulated by input substitution
Solution: add MACs

2) $z \leftarrow \text{Open}(A,[z])$:

- Bob sends $z_B, m_B$
- Alice outputs
  
  - $z = z_A + z_B$ if $m_B = z_B \Delta A + k_A$
  - “abort” otherwise

- **Solution**: Enhance representation $[x]$
  
  - $[x] = ( (x_A, k_A, m_A), (x_B, k_B, m_B) )$ s.t.
  
  - $m_B = x_B \Delta A + k_A$ (symmetric for $m_A$)
  - $\Delta A, \Delta B$ is the same for all wires.
Linear representation

• Given
  - \([x] = ( (x_A, k_{Ax}, m_{Ax}), (y_B, k_{Bx}, m_{Bx}) )\)
  - \([y] = ( (y_A, k_{Ay}, m_{Ay}), (y_B, k_{By}, m_{By}) )\)
  - Compute \([z] = (\)
    \[
    \begin{align*}
    z_A &= x_A + y_A, & k_{Az} &= k_{Ax} + k_{Ay}, & m_{Az} &= m_{Ax} + m_{Ay}, \\
    z_B &= x_B + y_B, & k_{Bz} &= k_{Bx} + k_{By}, & m_{Bz} &= m_{Bx} + m_{By}.
    \end{align*}
    \]
  \)

• And \([z]\) is in the right format since...
  \[
  m_{Bz} = (m_{Bz} + m_{By}) = (k_{Ax} + x_B \Delta_A) + (k_{Ay} + y_B \Delta_A)
  = (k_{Ax} + k_{Ay}) + (x_B + y_B) \Delta_A = k_{Az} + z_B \Delta_A
  \]
Recap

1. **Output Gates:**
   - Exchange shares and MACs
   - Abort if MAC does not verify

2. **Input Gates:**
   - Get a random \([r]\) from *trusted dealer*
   - \(r \leftarrow \text{Open}(A,[r])\)
   - Alice sends Bob \(d=x-r\),
   - Compute \([x]=[r]+d\)

Allows simulator to extract \(x^* = r+d^*\)
Recap

1. **Addition Gates:**
   - Use linearity of representation to compute
     \[ [z] = [x] + [y] \]

2. **Multiplication gates:**
   - Get a random triple \([a][b][c]\) with \(c = ab\) from
   - \(e \leftarrow \text{Open}([a]+[x])\), \(d \leftarrow \text{Open}([b]+[y])\)
   - Compute \([z] = [c] + a[y] + b[x] - ed\)
Final remarks

• Size of MACs

• Lazy MAC checks
Size of MACs

1. Each party must store a mac/key pair for each other party
   – quadratic complexity! 😞
   – SPDZ for linear complexity.

2. MAC is only as hard as guessing key!
   \[ k \text{ MACs in parallel give security } \frac{1}{|F|^k} \]
   – In TinyOT $F=\mathbb{Z}_2$, then MACs/Keys are $k$-bit strings
   – MiniMACs for constant overhead
Lazy MAC Check

\[ z^* \]

\[ [w + e] \]

\[ +e \]
Recap of Part 1

• Two protocols “in the trusted dealer model”
  – One Time-Truth Table
    • Storage $\exp(\text{input size})$ 😞
    • Communication $O(\text{input size})$ 😊
    • 1 round 😊
  – (SPDZ)/BeDOZa/TinyOT online phase
    • Storage linear #number of AND gates
    • Communication linear #number of AND gates
    • #rounds = depth of the circuit
  – ...and add enough MACs to get active security
Recap of Part 1

• To do secure computation is enough to precompute enough random multiplications!

• If no semi-trusted party is available, we can use cryptographic assumption (next)
Plan for the next 3 hours...

- **Part 1: Secure Computation with a Trusted Dealer**
  - Warmup: One-Time Truth Tables
  - Evaluating Circuits with Beaver’s trick
  - MAC-then-Compute for Active Security
- **Part 2: Oblivious Transfer**
  - OT: Definitions and Applications
  - Passive Secure OT Extension
  - OT Protocols from DDH (Naor-Pinkas/PVW)
- **Part 3: Garbled Circuits**
  - GC: Definitions and Applications
  - Garbling gate-by-gate: Basic and optimizations
  - Active security 101: simple-cut-and choose, dual-execution
Circuit Evaluation (Online phase)

3) Multiplication?

How to compute $[z]=[xy]$?

Alice, Bob should compute

$$z_A + z_B = (x_A + x_B)(y_A + y_B)$$

$$= x_A y_A + x_B y_A + x_A y_B + x_B y_B$$

Alice can compute this

Bob can compute this
Part 2: Oblivious Transfer

• OT: Definition, Applications (Gilboa’s protocol)

• OT Protocols from DDH (Naor-Pinkas/PVW)

• Passive Secure OT Extension
• Receiver does not learn $m_{1-b}$
• Sender does not learn $b$
1-2 OT

Receiver

Sender

\[ m_b = (1-b) m_0 + b m_1 \]

\[ m_b = m_0 + b (m_1 - m_0) \]
1-n OT

Receiver

1-n OT

Sender

$i$

$m_i$

$m_1, \ldots, m_n$
2PC via 1-n OT

Receiver

Sender

1-n OT

\[ f(x, y) \]

\[ f(1, y), \ldots, f(n, y) \]
Oblivious Transfer

= bit multiplication

\[ ab + c = (c, a+c) \]
n OTs = Arith. Multiplication

Receiver
\[ b = (b_0, b_1, \ldots, b_{n-1}) \]

Sender
\[ a \text{ (n bit number)} \]
\[ c_0 + \ldots + c_{n-1} = c \]

\[ d_i = a(2^i b_i) + c_i \]

\[ d_0 + \ldots + d_{n-1} = a(b_0 + 2b_1 + \ldots + 2^{n-1}b_{n-1}) + (c_0 + \ldots + c_{n-1}) = ab + c \]
Part 2: Oblivious Transfer

- OT definition, applications (Gilboa’s protocol)
- OT Protocols from DDH (Naor-Pinkas/PVW)
- Passive Secure OT Extension (IKNP03)
Passive Secure OT

Receiver(b)

pk_b \leftarrow G(sk)

pk_{1-b} \leftarrow \text{Rand}()

Sender(m_0,m_1)

(pk_0,pk_1)

c_0 = E(pk_0,m_0), c_1 = E(pk_1,m_1)

m_b = D(sk,c_b)

Receiver privacy: Real pk \approx "random" pk

Sender privacy: encryption is secure (Alice does not have sk)
Malicious

Receiver(b)

Sender(m₀,m₁)

pk₀ ← G(sk₀)
pk₁ ← G(sk₁)

(pk₀,pk₁)

c₀ = E(pk₀,m₀), c₁ = E(pk₁,m₁)

m₀ ← D(sk₀, c₀)
m₁ ← D(sk₁, c₁)
Active Secure OT

Receiver\((b)\)

Sender\((m_0,m_1)\)

\[ \text{mpk} \leftarrow f(\text{crs,sk,b}) \]

\[ c_0 = E(pk_0,m_0), \ c_1 = E(pk_1,m_1) \]

\[ m_b = D(sk,c_b) \]

Keys are correlated, Receiver cannot learn the sk for both.
Naor-Pinkas OT
(a la Chou-Orlandi)

Receiver(b)

Sender(m₀,m₁)

mpk = g^{skb}

From
pk₀ = g^{sk₀}
pk₁ = g^{sk₁}

h = pk₀/pk₁

→
h = g^{sk₀-sk₁}

c₀ = E(pk₀, m₀), c₁ = E(pk₁, m₁)

Encryption is ElGamal

crs = h (single group element)
Part 2: Oblivious Transfer

• OT definition, applications (Gilboa’s protocol)

• OT Protocols from DDH (Naor-Pinkas/PVW)

• Passive Secure OT Extension (IKNP03)
Efficiency

• *Problem*: OT requires public key primitives, inherently efficient
The Crypto Toolbox

Weaker assumption  Stronger assumption

OTP >> SKE >> PKE >> FHE >> Obfuscation

More efficient

Less efficient
Efficiency

• **Problem**: OT requires public key primitives, inherently efficient

• **Solution**: OT extension
  – Like hybrid encryption!
  – Start with few (expensive) OT based on PKE
  – Get many (inexpensive) OT using only SKE
WARMUP: USEFUL OT PROPERTIES
Short OT $\rightarrow$ Long OT

$\text{Receiver}$

$\text{Sender}$

$\mathbf{(u_0, u_1) = (prg(k_0) + m_0, prg(k_1) + m_1)}$

$m_b = \prg(k_b) + u_b$

$k_0, k_1$

$b$

$k_b$

$m_0, m_1$

poly(k)-bit strings

k-bit strings
Random OT = OT

\[ (x_0, x_1) = ((r_0 + m_0), (r_1 + m_1)) \]

If \( b = c \)

\[ m_b = r_c + x_b \]
Random OT = OT

Exercise: check that it works!
(R)OT is symmetric

\[ c = s_0 + s_1 \]
\[ z = s_0 \]
\[ r_0 = y \]
\[ r_1 = b + r_0 \]

No communication!

Exercise: check that it works
OT Extension

• OT pro(v/b)ably requires public-key primitivies

  – OT extension ≈ hybrid encryption

  – Start from k “real” OTs

  – Turn them into poly(k) OTs using only few symmetric primitives per OT
OT Extension, Pictorially

Remember:
OT stretching
(see “Short OT → Long OT” slide earlier)
Condition for OT extension

\[ X_1 \oplus c = X_0 \]

Remember: “Random OT → OT”

Problem for active security!
OT Extension, Pictorially

\[ X_1 \oplus b_1 \cdot c \]

\[ U \]

\[ k \]

1-2 CROTs

\[ k \]

[\text{"Correlated OTs"}]
OT Extension, Pictorially

\[(b \otimes c)_{ij} = b_i \cdot c_j\]
OT Extension, Turn your head!

\[ U \oplus X = b \quad c \]

\[ U \times C = c \]

\[ b \times c \]
OT Extension, Pictorially

\[ U \rightarrow b \rightarrow \text{1-2 CROT}s \rightarrow X \rightarrow c \]

\[ n = \text{poly}(k) \]

\[ k \]
OT Extension, Pictorially

\[
\begin{align*}
\sigma \subset & \quad n \\
\text{CROTs} & \quad n \\
1-2 & \\
\end{align*}
\]
Break the correlation!

\[ Y_0 = H \]

\[ Y_1 = H \]

\[ b \oplus \]
Breaking the correlation

- Using a **correlation robust hash function** $H$ s.t.
  1. $\{a_0, \ldots, a_n, H(a_0 + r), \ldots, H(a_n + r)\} \parallel (a_i's, r \text{ random})$
  2. $\{a_0, \ldots, a_n, b_0, \ldots, b_n\} \parallel (a_i's, b_i's \text{ random})$

  are **computationally indistinguishable**
OT Extension, Pictorially

\[ \text{n=poly}(k) \]

\[ 1\text{-}2\text{ ROTs} \]
Recap

0. Strech $k$ OTs from $k$- to poly($k$)=$n$-bitlong strings

1. Send correction for each pair of messages $x_i^0, x_i^1$
   s.t. $x_i^0 \oplus x_i^1 = c$

2. Turn your head (S/R swap roles)

3. The bits of $c$ are the new choice bits

4. Break the correlation: $y_j^0 = H(u_j)$, $y_j^1 = H(u_j \oplus b)$
   • Not secure against active adversaries
Recap of Part 2

• OT: building block for 2PC
  – Requires PKE 😞
  – OT Extension (using only SKE) 😊
  – Can be combined with protocols from part 1 for 2PC without a trusted dealer (using computational assumptions) 😊
  – \#rounds = depth of the circuit 😳
Coming up next...

- OT + Garbled Circuits → **Constant round 2PC!**

...aka layman fully-homomorphic encryption
Plan for the next 3 hours...

- **Part 1: Secure Computation with a Trusted Dealer**
  - Warmup: One-Time Truth Tables
  - Evaluating Circuits with Beaver’s trick
  - MAC-then-Compute for Active Security

- **Part 2: Oblivious Transfer**
  - OT: Definitions and Applications
  - Passive Secure OT Extension
  - OT Protocols from DDH (Naor-Pinkas/PVW)

- **Part 3: Garbled Circuits**
  - GC: Definitions and Applications
  - Garbling gate-by-gate: Basic and optimizations
  - Active security 101: simple-cut-and choose, dual-execution
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 $z = f(x)$
Passive Constant Round 2PC (Yao)

Alice(x)

[Z] ← Ev([F],[X],[Y])

z = De(d,[Z])

2PC (OT)

Bob(y)

([F],e,d) ← Gb( f,r )

[Y] ← En(e,y)

x

[F], [Y], d

e
Passive Constant Round 2PC (Yao)

Alice(x)

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

2PC (OT)

Bob(y)

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

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

Alice(x)

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

Bob(y)

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

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

Exist Sim s.t. ([F],[X],d) ~ Sim(f,f(x))

Or even less/no info about 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

\[ \begin{align*}
  z^* & \rightarrow [w+e] \\
  & \rightarrow [w] \\
 & \rightarrow w \\
 & \rightarrow +e
\end{align*} \]
PROJECTIVE SCHEMES:
CIRCUIT BASED GARBLING/EVALUATIONS
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, internal and output wires*

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

- 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

\[ [X] = \text{En}(e, x) \]
- \( e = \{ K^i_0, K^i_1 \} \)
- \( x = \{ x_1, \ldots, x_n \} \)
- \( [X] = \{ K^1_{x_1}, \ldots, K^n_{x_n} \} \)

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

- Parse \([X] = \{K^1, ..., K^n\}\)
- Parse \([F] = \{gg^i\}\)
- For each gate \(i\) compute
  - \(K \leftarrow \text{Ev}(gg^i,L,R)\)
- Output
  - \(Z\)
INDIVIDUAL GATES GARBLING/EVALUATION
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)

- NAND gate

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Yao Gate Garbling (2)

\[
\begin{array}{ccc}
L & R & K \\
L_0 & R_0 & K_1 \\
L_0 & R_1 & K_1 \\
L_1 & R_0 & K_1 \\
L_1 & R_1 & K_0 \\
\end{array}
\]

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

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 = H(L_0,R_0) \oplus K_1$</td>
</tr>
<tr>
<td>$C_2 = H(L_0,R_1) \oplus K_1$</td>
</tr>
<tr>
<td>$C_3 = H(L_1,R_0) \oplus K_1$</td>
</tr>
<tr>
<td>$C_4 = H(L_1,R_1) \oplus K_0$</td>
</tr>
</tbody>
</table>

• Encrypt the output key with the input keys
Yao Gate Garbling (4)

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

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

### Table

<table>
<thead>
<tr>
<th>C</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>$H(L_0, R_0) \oplus (K_1, 0^k)$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$H(L_0, R_1) \oplus (K_1, 0^k)$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$H(L_1, R_0) \oplus (K_1, 0^k)$</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$H(L_1, R_1) \oplus (K_0, 0^k)$</td>
</tr>
</tbody>
</table>

### Diagram

![Yao Gate Diagram](image)

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

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

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

- Evaluator learns nothing else:
  - Encryption + permutation

\begin{align*}
\text{gg (permuted)} \\
C_1 &= H(L_0, R_0) \oplus (K_1, 0^k) \\
C_2 &= H(L_0, R_1) \oplus (K_1, 0^k) \\
C_3 &= H(L_1, R_0) \oplus (K_1, 0^k) \\
C_4 &= H(L_1, R_1) \oplus (K_0, 0^k)
\end{align*}
GARBLING OPTIMIZATIONS:
POINT-AND-PERMUTE
**Point-and-permute**

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

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

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

\[
\begin{array}{|c|}
\hline
C \\
\hline
C_1 = H(L_0,R_0) \oplus (K_{g(0,0)}, r \oplus g(0,0)) \\
C_2 = H(L_0,R_1) \oplus (K_{g(0,1)}, r \oplus g(0,1)) \\
C_3 = H(L_1,R_0) \oplus (K_{g(1,0)}, r \oplus g(1,0)) \\
C_4 = H(L_1,R_1) \oplus (K_{g(1,1)}, r \oplus g(1,1)) \\
\hline
\end{array}
\]

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

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 = H(L_p, R_q) \oplus (K_{g(p,q)}, r \oplus g(p,q))$</td>
</tr>
<tr>
<td>$C_2 = H(L_p, R_{!q}) \oplus (K_{g(p,!q)}, r \oplus g(p,!q))$</td>
</tr>
<tr>
<td>$C_3 = H(L_{!p}, R_q) \oplus (K_{g(!p,q)}, r \oplus g(!p,q))$</td>
</tr>
<tr>
<td>$C_4 = H(L_{!p}, R_{!q}) \oplus (K_{g(!p,!q)}, r \oplus g(!p,!q))$</td>
</tr>
</tbody>
</table>

- Permute rows using $p,q$
Point-and-permute Evaluation

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

• (K,r)=C’\_2\cdot{u+v} \oplus H(L,R)

• Output is correct:
  – (Check permutation)

• Privacy:
  – u=p\oplus a, v=q\oplus b
  – p,q are “one time pads” for a,b

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1 = H(L_p,R_q) \oplus (K_{g(p,q)}, r \oplus g(p,q))</td>
</tr>
<tr>
<td>C_2 = H(L_p,R_!q) \oplus (K_{g(p,!q)}, r \oplus g(p,!q))</td>
</tr>
<tr>
<td>C_3 = H(L_!p,R_q) \oplus (K_{g(!p,q)}, r \oplus g(!p,q))</td>
</tr>
<tr>
<td>C_4 = H(L_!p,R_!q) \oplus (K_{g(!p,!q)}, r \oplus g(!p,!q))</td>
</tr>
</tbody>
</table>
GARBLING OPTIMIZATIONS:
SIMPLE GARbled ROW REDUCTION
Point-and-permute

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

\[
\begin{align*}
(gg, Z_0, Z_1) & \leftarrow Gb(g, L_0, L_1, R_0, R_1) \\
(Z_{g(a,b)}) & \leftarrow Ev(gg, L_a, R_b)
\end{align*}
\]
Garbling a Circuit: ([F], e, d) ← 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
  - $(gg, K_0^i, K_1^i) ← Gb(g, L_0, L_1, R_0, R_1)$

- 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)

<table>
<thead>
<tr>
<th>C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1 = H(L_0,R_0) \oplus K_1$</td>
<td></td>
</tr>
<tr>
<td>$C_2 = H(L_0,R_1) \oplus K_1$</td>
<td></td>
</tr>
<tr>
<td>$C_3 = H(L_1,R_0) \oplus K_1$</td>
<td></td>
</tr>
<tr>
<td>$C_4 = H(L_1,R_1) \oplus K_0$</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_1 = H(L_0, R_0)$ (C₁=0&lt;sup&gt;k&lt;/sup&gt;)</td>
</tr>
<tr>
<td>$C_2 = H(L_0, R_1) \oplus K_1$</td>
</tr>
<tr>
<td>$C_3 = H(L_1, R_0) \oplus K_1$</td>
</tr>
<tr>
<td>$C_4 = H(L_1, R_1) \oplus K_0$</td>
</tr>
</tbody>
</table>

- 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!)
GARBLING OPTIMIZATIONS:
FREE XOR
Free-XOR

- **Problem**: in BeDOZa linear gates are for free. What about GC?

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

\[
\begin{align*}
L_0 &= L_0 \oplus \Delta \\
L_1 &= L_0 \oplus \Delta \\
R_0 &= R_0 \oplus \Delta \\
R_1 &= R_0 \oplus \Delta \\
Z_0 &= Z_0 \oplus \Delta \\
Z_1 &= Z_0 \oplus \Delta
\end{align*}
\]

\[
\begin{align*}
(gg, Z_0) &\leftarrow Gb(g, L_0, R_0, \Delta) \\
(Z_{g(a,b)}) &\leftarrow Ev(gg, L_a, R_b)
\end{align*}
\]
Garbling a Circuit : \(( [F], e, d) \leftarrow Gb(f) \)

- 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 \( i \)
Garbling non-linear gates

• Like before, but requires “circular security assumption”
  – (Compatible with GRR and p&P)

• Example for AND gate
  – Evaluator sees

\[
\begin{align*}
\mathbf{L}_0, \mathbf{R}_0, \mathbf{K}_0, \\
\text{H}(\mathbf{L}_0 \oplus \Delta, \mathbf{R}_0 \oplus \Delta) \oplus \mathbf{K}_0 \oplus \Delta
\end{align*}
\]

  – And should not be able to compute \( \Delta \) !
Garbling/Evaluating XOR Gates

\[
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)
\]

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 \( Z_{a \oplus b} = L_a \oplus R_b \)

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

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

Alice

Bob

([F], e, d) ← Gb( f, r )
[Y] ← En(e, y)

OT

X

[F], [Y], d

x

e

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)

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

Bob(y)

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

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

Alice(x)

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

Bob(y)

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

Insecurity 2 (selective failure)
Insecurity 2 (selective failure)

Alice(x)

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

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

Bob(y)

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

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

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

x = 1 \rightarrow z^* = abort
SIMPLE TRICKS FOR ACTIVE SECURITY
Cut-And-Choose
2PC, simple cut-and-choose

Alice

If $\text{Gb}(f, r_j) \neq [F]_j$
abort

else

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

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

Bob

$([F]_i, e_i, d_i) \leftarrow \text{Gb}(f, r_i)$

$[Y]_j \leftarrow \text{En}(e_j, y)$

$x \leftarrow \text{OT}$

$[X_1], [X_2]$}

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

$\text{rand } j$

$r_j, [Y]_j$
2PC, simple cut-and-choose

If $G_b(f, r_j) \neq [F]_j$
 abort

else

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

$z = De(d_j, [Z]_j)$

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

• “Real cut-and-choose”
  – Use O(k) circuits, get security $2^{-k}$
  – Requires more complex techniques
Delegation via GC
Application 1: Delegation via GC

Preprocessing: The client work is proportional to $|f|$

Online: The client can delegate comp. The work of the client is independent of $|f|$
Application 1: Delegation via GC

Authenticity:
For all PPT A
\[ [Z^*] \leftarrow A([F],[X]), \]
then
\[ \text{De}([Z^*],d) \text{ is } f(x) \text{ or } \bot \]

Preprocessing:
The client work is proportional to \(|f|\)

Online:
The client can Delegate comp.
The work of the client is independent of \(|f|\)
Garbled Circuits: Authenticity

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

OR

\[ z^* = \text{abort} \]
Dual Execution
Alice

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

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

\([X] \leftarrow \text{En}(e,x)\)

Bob

\([F],[Y]\)

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

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

\([Z],d\)

De(d,[Z])

z/abort

De(d,[Z])

z/abort
Alice*: Authenticity → [Z] is the right output!

Bob: ([F], e, d) ← Gb( f, r )
[Y] ← En(e, y)

De(d, [Z]) == De(d, [Z*])

z/abort

f(x, y) or abort
Alice*

```
OT

[F], [Y]

x
[X]

e

OT

[G], [X]

[Z*]

De(d,[Z*])

z/abort

De(d,[Z])
```

Bob

```
([F],e,d) ↦ Gb(f,r)

[Y] ↦ En(e,y)

(Z*) ↦ Ev([G],[X],[Y])

Selective failure

[Z*] = [Z] iff y = 0

z/abort

1 bit leakage
```
Recap: Garbled Circuits

• Garbled circuits: allow to evaluate *encrypted functions* on *encrypted inputs*
  – With properties like *privacy, authenticity*, etc.

• Applications: *constant-round 2PC*

• Different techniques for garbling gates
  – Efficiency vs. Assumptions

• Active security
  – How to check that the *right function* is garbled?
  – Cut-and-choose and other tricks...
Want more?

- Cryptographic Computing – Foundations
  - [http://orlandi.dk/crycom](http://orlandi.dk/crycom)
  - Programming & Theory Exercises
  - Will be happy to answer questions by mail!