Secure multiparty computation - Part III

Josef Barta

Autumn School of Algfebra, 2014

Boolean circuit [1]

Definition

Let B be a basis. A *Boolean circuit* over B with n inputs and m outputs is a tuple $C = (V, E, \alpha, \beta, \omega)$, where (V, E) is a finite directed acyclic graph, $\alpha\colon E \to \mathbb{N}$ is an injective function, $\beta\colon V \to B \cup \{x_1, ..., x_n\}$, and $\omega\colon V \to \{y_1, ..., y_m\} \cup \{*\}$, such that the following conditions hold:

- If $v \in V$ has in-degree 0, then $\beta(v) \in \{x_1, ..., x_n\}$ or $\beta(v)$ is a 0-ary Boolean function (i.e. a Boolean constant) from B.
- If $v \in V$ has in-degree $d > 0$, then $\beta(v)$ is a d-ary Boolean function from B or a family of Boolean functions from B.
- For every i, $1 \leq i \leq n$, there is at most one node $v \in V$ such that $\beta(v) = x_i$.
- For every i, $1 \leq i \leq m$, there is at most one node $v \in V$ such that $\omega(v) = y_i$.

Boolean circuit [1]

If $v \in V$ has in-degree d_0 and out-degree d_1, then we say: v is a *gate* in C with *fan-in* d_0 and *fan-out* d_1. If v is a gate in C then we also write $v \in C$ instead of $v \in V$. If $e = (u, v) \in E$ then we say: e is a wire in C; more specifically we say that e is an input wire of v and an output wire of u; and that u is a *predecessor gate* of v. If $\beta(v) = x_i$ for some i then v is an *input gate* or *input node*. If $\omega(v) \neq *$ then v is an *output gate* or *output node*.

AND and OR gate

Tables with values for respective gate:

u	v	$w = u$ AND v	$w = u$ OR v
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Example

Garbled circuit [3]

Definition

Let C be a Boolean circuit that receives two inputs $x, y \in \{0,1\}^n$ and outputs $C(x,y) \in \{0,1\}^n$ (for simplicity, we assume that the input and output have the same length, n, as well as the security parameter). Each wire has a random value assigned as key for one and zero respectively and every gate is encrypted, so that given one key for each input wire, we can compute the appropriate key on the output wire. We also assume that if a circuit output wire comes from a gate g, then gate g has no wires that are input to other gates.

Garbled AND gate [2]

Table for an AND gate with garbled values and for a garbled AND (gAND) gate:

$g(u)$	$g(v)$	$g(w) = g(u)$ AND $g(v)$	$g(w) = g(u)$ gAND $g(v)$
k_u^0	k_v^0	k_w^0	$E_{k_u^0}(E_{k_v^0}(k_w^0))$
k_u^0	k_v^1	k_w^0	$E_{k_u^0}(E_{k_v^1}(k_w^0))$
k_u^1	k_v^0	k_w^0	$E_{k_u^1}(E_{k_v^0}(k_w^0))$
k_u^1	k_v^1	k_w^1	$E_{k_u^1}(E_{k_v^1}(k_w^1))$

Garbled OR gate [2]

Table for an OR gate with garbled values and for a garbled OR (gOR) gate:

$g(u)$	$g(v)$	$g(w) = g(u)$ OR $g(v)$	$g(w) = g(u)$ gOR $g(v)$
k_u^0	k_v^0	k_w^0	$E_{k_u^0}(E_{k_v^0}(k_w^0))$
k_u^0	k_v^1	k_w^1	$E_{k_u^0}(E_{k_v^1}(k_w^1))$
k_u^1	k_v^0	k_w^1	$E_{k_u^1}(E_{k_v^0}(k_w^1))$
k_u^1	k_v^1	k_w^1	$E_{k_u^1}(E_{k_v^1}(k_w^1))$

Output translation [2]

For each output wire we get a decoding table:

$$[(0, k_w^0), (1, k_w^1)]$$

Constructing a garbled circuit [2]

- given a Boolean circuit
  - assign garbled values to all wires
  - construct garbled gates using the garbled values
- central property
  - given a set of garbled values, one for each input wire, we can compute the entire circuit and obtain garbled values for the output wires
  - with a translation table for the output wires we can obtain output
  - nevertheless, nothing but the output is learned

Example

Computing a garbled circuit [2]

- even with a wrong key, symmetric encryption can return a seemingly correct result
- in the next few slides we present two possibilities how to ensure correctness
  - Option 1: PRF-based encryption with a redundant block of zeros
  - Option 2: signal bit to show which ciphertext to decrypt

Computing a garbled circuit - Option 1 [2]

- $E_k(m) = [r, F_k(r) \oplus (m||0^n)]$
- since F is a PRF, the probability of obtaining n consecutive zeros is negligible

Computing a garbled circuit - Option 2 [2]

- for every wire a random control bit is chosen
- the control bits indicate which keys are to be used in order to decrypt the output

$$
\begin{aligned}
(0,0) &\rightarrow E_{k_u^1}(E_{k_v^0}(k_w^0||1)) \\
(1,1) &\rightarrow E_{k_u^0}(E_{k_v^1}(k_w^0||1)) \\
(0,1) &\rightarrow E_{k_u^1}(E_{k_v^1}(k_w^1||0)) \\
(1,0) &\rightarrow E_{k_u^0}(E_{k_v^0}(k_w^0||1))
\end{aligned}
$$

The advantage of this option is that the circuit requires just two decryptions per gate, instead of an average of 5 otherwise.

Yao's protocol [2]

- 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 $w_1, ..., w_n$ be the input wires of P_1 and $w_{n+1}, ..., w_{2n}$ be the input wires of P_2
- P_1 sends P_2 the strings $k_1^{x_1}, ..., k_n^{x_n}$
- P_1 and P_2 run n oblivious transfers 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

Proof of security - P1 corrupted [3]

- P_1 sees only the messages from the oblivious transfer and the message sent by P_2 at the end
- by the correctness of the construction of the garbled circuit, P_2 obtains the correct output $f(x, y)$, except with negligible probability
- $\Rightarrow$ P_1 receives at the end of the real execution a message that almost certaintly equals $f(x, y)$

$\Rightarrow$

- a simulator that is given $(x, f(x, y))$ can simulate the complete view of P_1:
  - simulates view of P_1 in the oblivious transfer
  - writes $f(x, y)$ at the end of the transcript

Proof of security - P2 corrupted [3]

- we construct a simulator S_2 that is given input $(y, f(x, y))$ and generates the view of P_2 in the protocol
- since P_2 expects to receive a garbled circuit, so S_2 has to generate such
  - the circuit must return $f(x, y)$ to P_2 when executed according to protocol instructions
  - but S_2 does not know x, so it cannot generate the circuit honestly (S_2 cannot without knowing x possibly know, which keys to hand to P_2)

Proof of security - P2 corrupted [3]

- $\Rightarrow$ the garbled circuit generated by S_2 is a fake one that always evaluates to $f(x, y)$, without any regard to the keys used
  - we achieve this by using gate tables in which all four entries encrypt the same key
  - $\Rightarrow$ the values of the input wires do not affect the value on the output wire

Proof of security - P2 corrupted - Hybrid argument [3]

- real execution of the oblivious transfer protocol is indistinguishable from a hybrid distribution $H_{OT}(x, y)$
- next we consider a series of hybrids $H_i(x, y)$ in which one gate at a time is replaced in the real garbled circuit by a fake one
  - $H_0(x, y)$ contains the real garbled circuit
    $\Rightarrow \; H_0(x, y) = H_{OT}(x, y)$
  - $H_{|C|}(x, y)$ contains the same fake circuit as constructed by S_2

Proof of security - P2 corrupted - Indistinguishability [3]

- by a standard hybrid argument, it follows, that a distinguisher between $H_0(x, y)$ and $H_{|C|}(x, y)$ can be used to distinguish between two successive hybrids
  - however, the security of the encryption scheme that is used for generating the gate tables ensures that neighboring hybrids are computationally indistinguishable
- thus we conclude that $H_0(x, y)$ is indistinguishable from $H_{|C|}(x, y)$, and so $S_2(y, f(x, y))$ is indistinguishable from the view of the real P_2 participating in the real protocol

Example

References


Heribert Vollmer: Introduction to Circuit Complexity: A Uniform Approach, Springer Science & Business Media, 1999.


Yehuda Lindell: The Yao construction and Its Proof of Security, Winter School on Secure Computation and Efficiency, Department of Computer Science, Bar-Ilan University, 2011.


Yehuda Lindell and Benny Pinkas: A Proof of Yao's Protocol for Secure Two-Party Computation, Cryptology ePrint Archive, Report 2004/175, 2004, `http://eprint.iacr.org/`.