FHE-MPC Notes

Lecturer: Nigel Smart
Scribe: Gareth Davies

Lecture #6

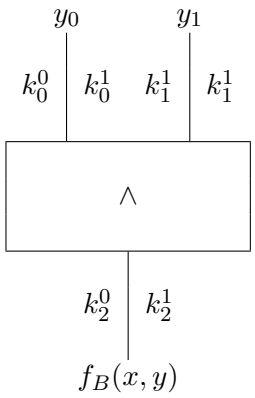
Aims. The main purpose of this lecture is to introduce **Yao circuits** and **two-party computation**. In order to understand the 2-party protocol of Yao it is necessary to introduce the notion of **oblivious transfer**.

Oblivious Transfer. In an oblivious transfer (OT) protocol, sender Alice provides two inputs, x_0 and x_1. Recipient Bob inputs a bit $b \in \{0, 1\}$. Bob then outputs x_b but does not know Alice's other input, whilst Alice does not ascertain which input Bob has received. In a two-party computation (2PC) protocol, Alice inputs x and Bob inputs y, and they each want to learn functions of these two variables, say $f_A(x, y)$ and $f_B(x, y)$ respectively, without learning each other's input.

Yao's crucial result was that an effective OT protocol implies the existence of a two-party computation protocol, so now we look at exactly what this is and why it works. In this setting we describe *Boolean circuits*, which consist of AND, OR, NAND, NOR, XOR and NOT gates with Boolean valued wires as inputs and outputs. Each wire comes out of a unique gate but a wire may fan-out, and thus we can represent the circuit as a collection of wires $W = \{w_1, ..., w_n\}$ and a collection of gates $G = \{g_1, ..., g_m\}$.

Two-Party Computation. Call Alice the *circuit creator* and Bob the *circuit evaluator*. If Bob wants to learn $f_B(x, y)$, then Alice creates circuits $C_F(y)$ using logic gates and sends the circuits to Bob. For every wire we create two secret keys. Yao's garbled circuit construction [5] is as follows:

- For each wire y_i two random cryptographic keys k_i^0 and k_i^1 are selected. The first key represents an encrpytion of 0 and the second represents encryption of 1.
- For each gate we compute a "garbled table" representing the function of the gate on these encrypted values. An initial table for an AND gate is shown below in Table 1.



AND gate with its corresponding wire keys.

a	b	
0	0	$\mathrm{Enc}_{k_0^0}(\mathrm{Enc}_{k_1^0}(k_2^0))$
0	1	$\mathrm{Enc}_{k_0^0}(\mathrm{Enc}_{k_1^1}(k_2^0))$
1	0	$\mathrm{Enc}_{k_0^1}(\mathrm{Enc}_{k_1^0}(k_2^1))$
1	1	$\mathrm{Enc}_{k_0^1}(\mathrm{Enc}_{k_1^1}(k_2^1))$

Table 1: Initial garbled circuit table for AND gate.

- The rows are shuffled, and then Bob picks a permutation $\pi\colon \{0,1\} \to \{0,1\}$ and inserts the new column into the table, then concatenates the "external wire value" π_a with the key and we drop the "internal value". A developed AND table with an example permutation is shown in Table 2.

$\pi_a(a)$	$\pi_b(b)$	
0	1	$\text{Enc}_{k_0^0}(\text{Enc}_{k_1^0}(k_2^0 \| \| 1))$
0	0	$\text{Enc}_{k_0^0}(\text{Enc}_{k_1^1}(k_2^0 \| \| 0))$
1	1	$\text{Enc}_{k_0^1}(\text{Enc}_{k_1^0}(k_2^1 \| \| 1))$
1	0	$\text{Enc}_{k_0^1}(\text{Enc}_{k_1^1}(k_2^1 \| \| 0))$

Table 2: Developed garbled circuit table.

- For each wire w_i the oblivious transfer runs as follows:
  - Alice inputs two secret keys k_i^0, k_i^1, B inputs $b \in \{0,1\}$,
  - B outputs k_i^b.
- Finally Alice sends the value of output wire keys k_2^0, k_2^1 so Bob can confirm his evaluation the circuits.

The main idea of the protocol is to keep the inputs private, apart from what could be deduced from the output of the function. Thus for an AND gate, if the output is 1 then of course both inputs must have been 1, whereas if he learns that the output is 0 he learns nothing about Alice's specific inputs.

2PC Schemes. One particular construction of the Yao framework is as follows [2]:

$$\text{Enc}^s_{k_1,k_2}(m) = m \oplus \text{KDF}^{[m]}(k_1, k_2, s), \tag{1}$$

where $[m]$ is the number of bits of the output, s is some string to identify the gate and output wire, and $\text{KDF} = H(k_1||s) \oplus H(k_2||s)$, where this is an AND gate.

Other constructions benefit from the *Free XOR Trick*, in which we do not need garbled circuits for XOR gates. In this method, instead of k_a^0 and k_a^1 we instead define $k_a^1 = k_a^0 \oplus R$, where R is some fixed randomness that is uniform throughout the gate. Thus the output keys will commute (with the randomness) and a garbled circuit will not need to be stored.

As is often the case, multiplication ($\wedge$) requires more computation than addition ($\oplus$). This scheme works if both Alice and Bob are honest, so now we look at how this may not be the case.

Cheating. Bob can cheat in the oblivious transfer step and by returning an invalid $g(x,y) \oplus \alpha$. Alice can cheat by sending invalid circuit data, OT data or output wires. To counteract cheating, we could use zero-knowledge proofs at each stage (this is far too time consuming to be practical), or use the *cut-and-choose* method [1]. In this framework, Alice generates s circuits $\{C_F^1, ..., C_F^s\}$, Bob commits to opening T of them, this process is repeated s times and $s - T$ circuits are deemed valid.

Progress. A computer system called Fairplay [3] was introduced in 2004, however it only dealt with honest parties. In 2008 the first implementation of systems involving honest parties was proposed [2].

At Asiacrypt 2009 the first computation of Yao circuits was introduced [4], and since this paper there have been numerous improvements in efficiency of the system. In this paper, the goal was to analyse a scheme in which Alice inputs a 128-bit AES key k, Bob inputs a 128-bit message m and Bob outputs a one-round AES encryption of m under k. The process involved 30k-45k gates, of which around 75% were XOR gates and the rest required garbled tables. The table below shows the comparative computation times at each step, for the honest party and malicious party case.

Alice	Bob	Runtime, Honest Adv.	Runtime, Malicious Adv.
Compute C_F		1	453
	$\xrightarrow{\text{send } C_F}$	1	276
	$\xleftrightarrow{OT}$	3	35
	Compute Result	2	350
		7 secs	**1114 secs**

Table 3: Runtimes for Different Stages of Protocol

Using a process known as inter-weaving (not doing the steps separately) it is possible to reduce the honest case to considerably less than seven seconds.

References

[1] Y. Lindell and B. Pinkas, *An efficient protocol for secure two-party computation in the presence of malicious adversaries*, Advances in Cryptology-EUROCRYPT 2007 (2007), no. 860, 52–78.

[2] Y. Lindell, B. Pinkas, and N. Smart, *Implementing two-party computation efficiently with security against malicious adversaries*, Security and Cryptography for Networks (2008), no. 781, 2–20.

[3] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, *Fairplay A Secure Two-Party Computation System*, Proceedings of the 13th conference on USENIX Security Symposium-Volume 13, USENIX Association, 2004, pp. 20–20.

[4] B. Pinkas, T. Schneider, N. Smart, and S. Williams, *Secure two-party computation is practical*, Advances in CryptologyASIACRYPT 2009 (2009), 250–267.

[5] A. Yao, *Protocols for secure computations*, Proceedings of FOCS (1982), 160–164.