Cryptography and Secure Two-Party Computation

Gabriel Bender

August 21, 2006

1 The Millionaire Problem and Secure Computation

In 1982, Andrew Yao proposed the following problem: Alice and Bob are millionaires, and the pair wishes to determine who has more money. However, neither person wishes to reveal his or her precise wealth to the other. [5]

One approach to this problem (dubbed the ideal solution) is for Alice and Bob to find a third party that they both trust. Then Alice tells the third party how much money she has, and Bob follows suit. Once both Alice and Bob have both revealed the necessary information to him, the third party provides both of them with an answer to their question. However, the use of such a trusted third party is not always practical. Therefore, our goal is to simulate the presence of such a third party using cryptography.

Secure two-party computation provides a generalization of this problem: Suppose we wish to compute the value of a function $F(A, B)$ where A is an input provided by Alice and B is provided by Bob. Assume furthermore that Alice does not wish to disclose her input to Bob, and that likewise, Bob wishes to conceal his input from Alice. How can they compute $F(A, B)$ together while observing these constraints?

2 Private and Public-Key Cryptography

Traditionally, cryptography has dealt with the secure exchange of information between two parties over an insecure channel. In other words, suppose Alice wishes to send a message to Bob, but an adversary Eve intercepts every message that either person sends. How can Alice send a message which can be deciphered by Bob but not by Eve?

The classical solution to this problem is for Alice and Bob to meet ahead of time and share a piece of information called a key. With the key, a message can be either encrypted or decrypted. Without the appropriate key, neither encryption nor decryption can be performed. Suppose that $E(\cdot)$ is the pair's encryption function and $D(\cdot)$ is the pair's decryption function. Then if Alice wishes to communicate a secret message m to Bob, she sends him the encrypted value $E(m)$. Bob then applies his decryption function to obtain $D(E(m)) = m$. On the other hand, since Eve does not have access to the key that Alice and Bob use, she can not extract any information about m even if she intercepts $E(m)$.

However, this method requires that Alice and Bob meet ahead of time in order to decide on a key. Public-key cryptography allows us to forgo this requirement. More specifically, with public-key cryptography, our adversary Eve is unable to extract any information about a message m even if she has access to both an encryption function $E(\cdot)$ and an encrypted message $E(m)$. However, the message can still be decrypted by a user with access to a separate decryption function $D(\cdot)$. If Alice wishes to send a message to Bob using public-key cryptography, the pair uses the following procedure:

- Bob randomly selects an encryption function E and corresponding decryption function D such that $D(E(m)) = m$. He sends the encryption function $E(\cdot)$ to Alice.

- Alice sends the encrypted message $E(m)$ to Bob.

- Bob decrypts the message by calculating $D(E(m)) = m$.

Since Eve knows only the encyption function $E(\cdot)$ and the value $E(m)$, she is unable to collect any further information about m.

3 Oblivious Transfer

One-out-of-two oblivious transfer solves the following problem: Alice possesses two messages m_0 and m_1. Bob selects $i \in \{0, 1\}$ and obtains the message of his choice, m_i, while collecting no information about the other message, m_{1-i}. Alice is not allowed to know which message Bob selected. [3]

Suppose that a and b are both strings in $\{0,1\}^n$. Then let $a \oplus b$ denote the component-wise XOR of a and b, with the following truth table:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

This is equivalent to addition modulo 2. Notice that $(x \oplus y) \oplus x = y \oplus (x \oplus x) = y$. Now, assume that for arbitrary $n \in \mathbb{N}$, Alice can generate an encryption function $E : \{0,1\}^n \to \{0,1\}^n$ that is one-to-one and onto, and a corresponding decryption function $D(\cdot)$, with $E(\cdot)$ and $D(\cdot)$ subject to the constraints of public-key cryptography. Oblivious transfer for two strings of length n can be accomplished with the following procedure:

- Alice generates an encryption function $E(\cdot)$ with corresponding decryption function $D(\cdot)$. She sends E to Bob and keeps D to herself.
- Bob generates two random strings x_0 and x_1 of length n. Let us assume that he wishes to obtain m_0. The case in which he obtains m_1 is entirely analogous.
- Bob sends the tuple $(E(x_0), x_1)$ to Alice.
- Alice applies the function D to both elements of Bob's tuple, obtaining the pair $(D(E(x_0)), D(x_1)) = (x_0, D(x_1))$.
- Alice XORs the first element of the tuple with m_0 and the second element with m_1. She sends Bob the resulting tuple $(x_0 \oplus m_0, D(x_1) \oplus m_1)$.
- Since Bob knows x_0, he is able to calculate $(x_0 \oplus m_0) \oplus x_0 = m_0$. However, he can not compute $D(x_1)$, and is therefore unable to obtain m_1 from this process.

Since the domain and range of the encryption functions are identical, it is impossible for Alice to determine whether Bob has sent her x_0 or $E(x_0)$. Consequently, Alice gains no information about whether Bob plans to obtain the value of m_0 or m_1 from the input that he sends her. And as we have already seen, Bob can only extract one of the two values m_0 and m_1.

It is worth noting that this example requires that both parties are willing to "play by the rules" of the oblivious transfer. For example, Bob could send Alice the tuple $(E(x_0), E(x_1))$ in the third step and obtain the values of both m_0 and m_1 from Alice. For the sake of simplicity, we shall assume that Alice and Bob are semi-honest, so that they are required to follow the steps of the protocol, even if they do additional calculations on the side. The following discussion of scrambled circuits makes the same assumptions about semi-honest behavior. In practice, it is possible to force parties to behave in a semi-honest manner. [2]

4 Scrambled Logic Gates

Now, let us consider the following problem:

$F : \{0,1\} \times \{0,1\} \to \{0,1\}$ is a boolean function that takes two arguments A and B as input. Alice supplies the value of A and Bob supplies the value of B. However, neither person wishes to disclose his or her own value to the other. In an ideal setup, Alice and Bob would secretly supply their inputs to a trusted third party, who would perform the necessary computation and inform both of them of the result. However, Alice and Bob can use cryptography to simulate the effect of asking a third party. In this procedure, one person (say Alice) is appointed the circuit-maker and the other (say Bob) is the circuit-executor. [3]

- Alice selects two (not necessarily distinct) random permutations ν_0 and ν_1 of the set $\{0,1\}$.
- Alice generates four random encryption functions $E_0^0, E_1^0, E_0^1,$ and E_1^1 with corresponding decryption functions $D_0^0, D_1^0, D_0^1,$ and D_1^1.
- Alice constructs the following truth table:

In 0 (Alice)	In 1 (Bob)	Encrypted Output
$\nu_0(0)$	$\nu_1(0)$	$E^0_{\nu_0(0)}(E^1_{\nu_1(0)}(F(0,0)))$
$\nu_0(0)$	$\nu_1(1)$	$E^0_{\nu_0(0)}(E^1_{\nu_1(1)}(F(0,1)))$
$\nu_0(1)$	$\nu_1(0)$	$E^0_{\nu_0(1)}(E^1_{\nu_1(0)}(F(1,0)))$
$\nu_0(1)$	$\nu_1(1)$	$E^0_{\nu_0(1)}(E^1_{\nu_1(1)}(F(1,1)))$

- Alice sends this truth table, along with the function $\nu_1(\cdot)$, to Bob.

The gate has now been constructed, and Bob is ready to evaluate it. Alice decides on an input α to F, and Bob decides on an input β. Then the pair follows this procedure:

- Alice sends the tuple $(\nu_0(\alpha), D^0_{\nu_0(\alpha)})$ to Bob.
- Bob uses oblivious transfer to request the decryption function $D^1_{\nu_1(\beta)}$. He then uses this knowledge to construct his own tuple $(\nu_1(\beta), D^1_{\nu_1(\beta)}(\cdot))$.
- Bob finds the unique row of the truth table in which Alice's input is $\nu_0(\alpha)$ and his own input is $\nu_1(\beta)$. The encrypted output for this row is $E^0_{\nu_0(\alpha)}(E^1_{\nu_1(\beta)}(F(\alpha, \beta)))$
- Bob applies both of his decryption keys to the entry to obtain the tuple

  $$D^1_{\nu_1(\beta)}(D^0_{\nu_0(\alpha)}(E^0_{\nu_0(\alpha)}(E^1_{\nu_1(\beta)}(F(\alpha, \beta))))) = F(\alpha, \beta)$$

  This is the unencrypted output of the gate. He passes the value along to Alice.

The only time Bob provides information to Alice about his input is when he requests the decryption key $D^1_{\nu_1(\beta)}(\cdot)$. However, this is done using an oblivious transfer, so that Alice can not obtain any information from Bob's requests.

There are two ways for Bob to gather information about Alice's input. The first is by analyzing the value $\nu_0(\alpha)$ to determine the value of α. However, ν_0 is a random permutation unknown to Bob, so he can not use his knowledge of $\nu_0(\alpha)$ to obtain any information about the true value of α. Alternatively, Bob could attempt to decrypt another row of the truth table to see how the result of the calculation would have changed if his or Alice's input was different. However, Bob possesses only two decryption functions, which together only allow him to decrypt a single row of the truth table. That row corresponds with the value of the function when Alice provides input α and Bob provides input β, so he gains no extra information that way.

It is worth noting that the function ν_1 is entirely superfluous in this example, since it is known by both Alice and Bob. However, it will simplify the notation for the more general case.

5 Secure Evaluation of Larger Circuits

In practice, it is extremely limiting to say that Alice and Bob must only work with boolean functions. One way of working around this is to expand the truth table to accommodate a larger set of input values. However, this requires that the truth table grow exponentially with the length of the input values. For example, if Alice and Bob each input a single n-bit binary string to a function F, then the truth table for F must contain 2^{2n} rows.

Instead, we introduce the notion of a circuit. A circuit consists of three parts:

- A set of initial input values $x_1, \ldots, x_m$
- A set of logic gates $F_1, \ldots, F_n : \{0,1\} \times \{0,1\} \to \{0,1\}$.
- A collection of wires specifying the inputs to the logic gates. An input to a logic gate may be one of the initial input values or the output of another logic gate. To avoid feedback, we require that each logic gate F_k may only take input from earlier gates $F_1, \ldots, F_{k-1}$.

We define the output of the entire circuit to be the output of the last logic gate, F_n.

Our goal is to extend the limited implementation for single gates from the previous section to the secure evaluation of entire circuits. Our plan is to use the output of one logic gate as the scrambled input to another. To do this, we create "scrambled tuples" that describe the inputs and outputs of logic gates. Each scrambled tuple takes the form $(\nu_i(x_i), D^i_{\nu_i(x_i)})(\cdot)$, where $\nu_i(\cdot)$ is a randomly selected permutation of $\{0,1\}$, x_i is the "true" unscrambled input or output of the gate, and $D^i_{\nu_i(x_i)}(\cdot)$ is a decryption function. The idea is that the scrambled output tuple of one gate is used as the scrambled input tuple for another. In this way, Bob is unable to determine any information about the unencrypted values of the gates because he does not know what the $\nu_i(\cdot)$ functions are, and he will only be able to access one row of in the truth table of each gate: the one corresponding to Alice's and Bob's inputs. He can possess at most two decryption keys for each gate, and will be limited by the same restrictions he used in the previous section. To convert a regular circuit into a scrambled circuit, Alice uses the following procedure, which includes a collection of "scrambled output values" denoted $S_1, S_2, \ldots$:

- She associates S_1 with with the input value x_1, S_2 with x_2, and so on, until she associates S_m with x_m.
- She associates S_{m+1} with the gate F_1, ..., S_{m+n} with F_n.
- She randomly generates a permutation $\nu_i(\cdot) : \{0,1\} \to \{0,1\}$ for each $1 \le i \le m+n$.
- She randomly generates two encryption keys E_0^i and E_1^i with corresponding decryption keys D_0^i and D_1^i for each $1 \le i \le m+n$.
- For each gate F_i with scrambled inputs S_x and S_y, Alice constructs the following truth table:

Input 1	Input 2	Encrypted Output
$\nu_x(0)$	$\nu_y(0)$	$E^x_{\nu_x(0)}(E^y_{\nu_y(0)}(F_i'(0,0)))$
$\nu_x(0)$	$\nu_y(1)$	$E^x_{\nu_x(0)}(E^y_{\nu_y(1)}(F_i'(0,1)))$
$\nu_x(1)$	$\nu_y(0)$	$E^x_{\nu_x(1)}(E^y_{\nu_y(0)}(F_i'(1,0)))$
$\nu_x(1)$	$\nu_y(1)$	$E^x_{\nu_x(1)}(E^y_{\nu_y(1)}(F_i'(1,1)))$

  In this case, we define $F_i'(p,q)$ to be the scrambled tuple $(\nu_i(t), D^i_{\nu_i(t)}(\cdot))$ where $t = F_i(p,q)$.

  This tuple is the scrambled output of gate F_i, as well as the scrambled value associated with S_{m+i}.

Now, Alice and Bob are ready to evaluate the circuit. Their strategy is to evaluate the scrambled values $S_1, \dots, S_{m+n}$ in that order. Alice begins by sending all of her truth tables to Bob. [2]

For each of the values $x_1, \dots, x_m$, they use the same technique they did in the single-circuit case. If Alice is responsible for supplying the value of x_i, she contributes the tuple $S_i = (\nu_i(x_i), D^i_{\nu_i(x_i)}(\cdot))$ without revealing either x_i or ν_i to Bob. If Bob is responsible for contributing the value, Alice first sends the function $\nu_i(\cdot)$ to Bob. Bob then uses oblivious transfer to obtain the decryption function $D^i_{\nu_i(x_i)}(\cdot)$ from Alice. Finally, Bob sets $S_i = (\nu_i(x_i), D^i_{\nu_i(x_i)}(\cdot))$.

Suppose that Bob wishes to evaluate gate F_i, given that he has already evaluated gates $F_1, \dots, F_{i-1}$. Call the input statements to this gate S_a and S_b. We may then assume that Bob has already obtained the scrambled tuples associated with those statements: $S_a = (\nu_a(t_a), D^a_{\nu_a(t_a)}(\cdot))$ and $S_b = (\nu_b(t_b), D^b_{\nu_b(t_b)}(\cdot))$, where t_a and t_b are the "true" values of the inputs a and b. Bob then has enough information to decrypt precisely one row of the truth table for the gate F_i. That row corresponds with the appropriate output of the gate when the first input value is t_a and Bob's value is t_b, which is precisely the row of the table Bob needs to proceed with his calculation. He can now use the scrambled tuple $(\nu_i(t_i), D^i_{\nu_i(t_i)}(\cdot))$ in his calculations for gates $F_{i+1}, \ldots, F_n$. When he finally obtains the scrambled output of F_n, he sends the value $\nu_{m+n}(t_{m+n})$ to Alice, who knows the value of the function ν and is therefore able to calculate the final output value t_{m+n} for the circuit.

As in the single-gate example, the only information that Alice obtains about Bob's inputs comes from the oblivious transfers through which he obtains the decryption keys $D^i_v(\cdot)$ for his initial inputs. By the nature of oblivious transfer, she can not obtain any information from this procedure.

And once again, Bob has two chances to obtain information about Alice's input. Every input to a truth table is labeled as a 0 or 1, but the actual values are concealed using randomly generated permutations. With the exception of the permutations associated with Bob's initial inputs, these functions are known only to Alice, so Bob can not gain any information about them this way. Alternatively, Bob could try to change inputs to his truth tables. But again, the truth tables are encrypted, so Bob can only access the "correct" row of the truth table for each gate in the circuit, so he gains no information that way either.

References

[1] O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2001.

[2] O. Goldreich. *Foundations of Cryptography: Basic Applications.* Cambridge University Press, 2004.

[3] O. Goldreich, S. Micali, and A. Wigderson. *How to play ANY mental game.* Annual ACM Symposium on Theory of Computing. Proceedings of the nineteenth annual ACM conference on Theory of Computing. Pages 218-229. http://portal.acm.org/. http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=28420

[4] 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.iapr.org/. http://citeseer.ist.psu.edu/lindell04proof.html

[5] Andrew Yao. *Protocols for Secure Computations (Extended Abstract).* 1982 University of Wisconsin Madison. 31 July. 2006. http://www.cs.wisc.edu/areas/sec/yao1982-ocr.pdf