Cryptography: An Introduction (3rd Edition)

Nigel Smart

CHAPTER 24

Commitments and Oblivious Transfer

Chapter Goals

- To present two protocols which are carried out between mutually untrusting parties.
- To introduce commitment schemes and give simple examples of efficient implementations.
- To introduce oblivious transfer, and again give simple examples of how this can be performed in practice.

1. Introduction

In this chapter we shall examine a number of more advanced cryptographic protocols which enable higher level services to be created. We shall particularly focus on protocols for

- commitment schemes,
- oblivious transfer.

Whilst there is a large body of literature on these protocols, we shall keep our feet on the ground and focus on protocols which can be used in real life to achieve practical higher level services. It turns out that these two primitives are in some sense the most basic atomic cryptographic primitives which one can construct.

Up until now we have looked at cryptographic schemes and protocols in which the protocol participants have been honest, and we are trying to protect their interests against an external adversary. However, in the real world often we need to interact with people who we do not necessarily trust. In this chapter we examine two types of protocol which are executed between two parties, for which each party may want to cheat in some way. In later chapters we shall present more complicated examples of similar protocols. The simplistic protocols in this chapter will form the building blocks on which more complicated protocols can be built.

We start by focusing on commitment schemes, and then we pass to oblivious transfer.

2. Commitment Schemes

Suppose Alice wishes to play 'paper-scissors-stone' over the telephone with Bob. The idea of this game is that Alice and Bob both choose simultaneously one of the set {`paper`,`scissors`,`stone`}. Then the outcome of the game is determined by the rules

- `Paper` wraps `stone`. Hence if Alice chooses `paper` and Bob chooses `stone` then Alice wins.
- `Stone` blunts `scissors`. Hence if Alice chooses `stone` and Bob chooses `scissors` then Alice wins.
- `Scissors` cut `paper`. Hence if Alice chooses `scissors` and Bob chooses `paper` then Alice wins.

If both Alice and Bob choose the same item then the game is declared a draw. When conducted over the telephone we have the problem that whoever goes first is going to lose the game.

One way around this is for the party who goes first to 'commit' to their choice, in such a way that the other party cannot determine what was committed to. Then the two parties can reveal

their choices, with the idea that the other party can then verify that the revealing party has not altered its choice between the commitment and the revealing stage. Such a system is called a commitment scheme. An easy way to do this is to use a cryptographic hash function as follows:

$$\begin{aligned} A &\longrightarrow B : h_A = H(R_A \| \texttt{paper}), \\ B &\longrightarrow A : \texttt{scissors}, \\ A &\longrightarrow B : R_A, \texttt{paper}. \end{aligned}$$

At the end of the protocol Bob needs to verify that the h_A sent by Alice is equal to $H(R_A\|\texttt{paper})$. If the values agree he knows that Alice has not cheated. The result of this protocol is that Alice loses the game since `scissors` cut `paper`.

Let us look at the above from Alice's perspective. She first commits to the value `paper` by sending Bob the hash value h_A. This means that Bob will not be able to determine that Alice has committed to the value `paper`, since Bob does not know the random value of R_A used and Bob is unable to invert the hash function. The fact that Bob cannot determine what value was committed to is called the concealing, or hiding property of a commitment scheme.

As soon as Bob sends the value `scissors` to Alice, she knows she has lost but is unable to cheat, since to cheat she would need to come up with a different value of R_A, say R'_A, which satisfied

$$H(R_A\|\texttt{paper}) = H(R'_A\|\texttt{stone}).$$

But this would mean that Alice could find collisions in the hash function, which for a suitable chosen hash function is believed to be impossible. Actually we require that the hash function is second-preimage resistant in this case. This property of the commitment scheme, that Alice cannot change her mind after the commitment procedure, is called binding.

Let us now study these properties of concealing and binding in more detail. Recall that an encryption function has information theoretic security if an adversary with infinite computing power could not break the scheme, whilst an encryption function is called computationally secure if it is only secure when faced with an adversary with polynomially bounded computing power. A similar division can be made with commitment schemes, but now we have two security properties, namely concealing and binding. One property protects the interests of the sender, and one property protects the interests of the receiver. To simplify our exposition we shall denote our our abstract commitment scheme by a public algorithm, $c = C(x, r)$ which takes a value x and some randomness r and produces a commitment c. To confirm a decomitment the commiter simply reveals the values of x and r. The receiver then checks that the two values produce the original commitment.

DEFINITION 24.1 (Binding). *A commitment scheme is said to be information theoretically (resp. computationally)* binding *if no infinitely powerful (resp. computationally bounded) adversary can win the following game.*

- *The adversary outputs a value c, plus values x and r which produce this commitment.*
- *The adversary must then output a value $x' \neq x$ and a value r' such that*

$$C(x, r) = C(x', r').$$

DEFINITION 24.2 (Concealing). *A commitment scheme is said to be information theoretically (resp. computationally)* concealing *if no infinitely powerful (resp. computationally bounded) adversary can win the following game.*

- *The adversary outputs two messages x_0 and x_1 of equal length.*
- *The challenger generates r at random and a random bit $b \in \{0, 1\}$.*
- *The challenger computes $c = C(x_b, r)$ and passes c to the adversary.*
- *The adversaries goal is to now guess the bit b.*

Notice, how this definition of concealing is virtually identical to our definition of indistinguishability of encryptions. A number of results trivially follow from these two definitions:

LEMMA 24.3. *There exists no scheme which is both information theoretically concealing and binding.*

PROOF. To be perfectly binding a scheme must be deterministic, since there needs to be a one-to-one relationship between the space of commitments and the space of committed values. But a deterministic scheme clearly will not meet the concealing definition. □

LEMMA 24.4. *Using the commitment scheme defined as*

$$H(R\|C),$$

for a random value R, the committed value C and some cryptographic hash function H, is at best

- *computationally binding,*
- *information theoretically concealing.*

PROOF. All cryptographic hash functions we have met are only computationally secure against preimage resistance and second-preimage resistance.

The binding property of the above scheme is only guaranteed by the second-preimage resistance of the underlying hash function. Hence, the binding property is only computationally secure.

The concealing property of the above scheme is only guaranteed by the preimage resistance of the underlying hash function. Hence, the concealing property looks like it should be only computationally secure. However, if we assume that the value R is chosen from a suitably large set, then the fact that the hash function should have many collisions works in our favour and in practice we should obtain something close to information theoretic concealing. On the other hand if we assume that H is a random oracle, then the commitment scheme is clearly information theoretically concealing. □

We now turn to three practical commitment schemes which occur in various protocols. All are based on a finite abelian group G of prime order q, which is generated by g. We let $h \in \langle g \rangle$, where the discrete logarithm of h to the base g is unknown by any user in the system. This latter property is quite easy to ensure, for example for a finite field $\mathbb{F}_p^*$, with q dividing $p-1$ we create g as follows (with a similar procedure being used to determine h):

- Pick a random $r \in \mathbb{Z}$.
- Compute $f = H(r) \in \mathbb{F}_p^*$ for some cryptographic hash function H.
- Set $g = f^{(p-1)/q} \pmod p$. If $g = 1$ then return to the first stage, else output (r, g).

This generates a random element of the subgroup of $\mathbb{F}_p^*$ of order q, with the property that it is generated verifiably at random since one outputs the seed r used to generate the random element.

Given g, h we define two commitment schemes, $B(x)$ and $B_a(x)$, to commit to an integer x modulo q, and one $E_a(x)$ to commit to an integer x modulo p.

$$\begin{aligned} B(x) &= g^x, \\ E_a(x) &= \left(g^a, x \cdot h^a\right), \\ B_a(x) &= h^x g^a, \end{aligned}$$

where a is a random integer modulo q. The scheme given by $B_a(x)$ is called Pedersen's commitment scheme. The value a is called the blinding number, since it blinds the value of the commitment x even to a computationally unbounded adversary. To reveal the commitments the user publishes the value x in the first scheme and the pair (a, x) in the second and third schemes.

LEMMA 24.5. *The commitment scheme $B(x)$ is information theoretically binding.*

PROOF. Suppose Alice having published

$$c = B(x) = g^x$$

wished to change her mind as to which element of $\mathbb{Z}/q\mathbb{Z}$ she wants to commit to. Alas, for Alice no matter how much computing power she has there is mathematically only one element in $\mathbb{Z}/q\mathbb{Z}$, namely x, which is the discrete logarithm of the commitment c to the base g. Hence, the scheme is clearly information theoretically binding. □

Note the Pederson commitment scheme does not meet our strong definition of security for the concealing property. If the space of values from which x is selected is large, then this commitment scheme could meet a weaker security definition related to a one-way like property.

LEMMA 24.6. *The commitment scheme $E_a(x)$ is information theoretically binding and computationally concealing.*

PROOF. This scheme is exactly ElGamal encryption with respect to a public key h. Note that we do not need to know the associated private key to use this as a commitment scheme. Indeed any semantically secure public key encryption scheme can be used in this way as a commitment scheme.

The underlying semantic security implies that the resulting commitment scheme is computationally concealing. Whilst the fact that the decryption is unique, implies that the commitment scheme is information theoretically binding. □

LEMMA 24.7. *The commitment scheme $B_a(x)$ is computationally binding and information theoretically concealing. That it is computationally binding only holds if the commiter does not know the discrete logarithm of h to the base g.*

PROOF. Now suppose Alice, after having committed to

$$b = B_a(x) = h^x g^a$$

wishes to change her mind, so as to commit to y instead. All that Alice need do is to compute

$$f = \frac{b}{h^y}.$$

Alice then computes the discrete logarithm a' of f to the base g. When Alice is now asked to reveal her commitment she outputs (a', y) instead of (a, x). Hence the scheme is at most computationally binding.

We can also show that if Alice, after having committed to

$$b = B_a(x) = h^x g^a$$

wishes to change her mind, then the only way she can do this is by computing the discrete logarithm of h to the base g. To see this first note that the value she changes her mind, say y, she must be able to decommit to. Hence, she must know the underlying randomness say b. Thus Alice knows x, y, a and b such that

$$h^x g^a = h^y g^b.$$

From which Alice can recover the discrete logarithm of h with respect to g in the standard manner.

Now suppose the recipient wishes to determine which is the committed value, before the revealing stage is carried out. Since, for a given value of b and every value of x, there is a value of a which makes a valid commitment, even a computationally unbounded adversary could determine no information. Hence, the scheme is information theoretically concealing. □

We end this section by noticing that the two discrete logarithm based commitment schemes we have given possess the homomorphic property:

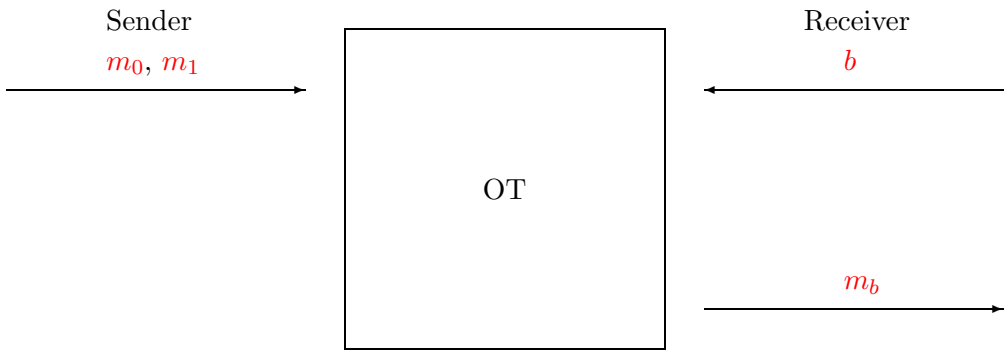
$$\begin{aligned} B(x_1) \cdot B(x_2) &= g^{x_1} \cdot g^{x_2} \\ &= g^{x_1+x_2} \\ &= B(x_1 + x_2), \\ B_{a_1}(x_1) \cdot B_{a_2}(x_2) &= h^{x_1} \cdot g^{a_1} \cdot h^{x_2} \cdot g^{a_2} \\ &= h^{x_1+x_2} \cdot g^{a_1+a_2} \\ &= B_{a_1+a_2}(x_1 + x_2). \end{aligned}$$

We shall use this homomorphic property when we discuss our voting protocol at the end of chapter 25.

3. Oblivious Transfer

We now consider another type of basic protocol called oblivious transfer, or OT for short. This is another protocol which is run between two distrusting parties, a sender and a receiver. In its most basic form the sender has two secret messages as input m_0 and m_1; the receiver has as input a single bit b. The goal of an OT protocol is that at the end of the protocol the sender should not learn the value of the receivers input b. However, the receiver should learn the value of m_b but should learn nothing about m_{1-b}. Such a protocol is often called a 1-out-of-2 OT, since the receiver learns one of the two inputs of the sender. Such a protocol can be visually pictured as in Figure 1. One can easily generalise this concept to an k-out-of-n OT, but as it is we will only be interested in the simpler case.

FIGURE 1. Pictorial Description of an 1-out-of-2 OT



We present a scheme which allows us to perform a 1-out-of-2 oblivious transfer of two arbitrary bit strings m_0, m_1 of equal length. The scheme is based on the following version of ElGamal (resp. DHIES). We take standard discrete logarithm based public/private key pair $(h = g^x, x)$, where g is a generator of cyclic finite abelian group G of prime order q. We will require a hash function H from G to bit strings of length n. Then to encrypt messages m of length n we compute, for a random $k \in \mathbb{Z}_q$

$$c = (c_1, c_2) = \left(g^k, m \oplus H(h^k)\right).$$

To decrypt we compute

$$c_2 \oplus H({c_1}^x) = m \oplus H(h^k) \oplus H(g^{kx}) = m.$$

Notice, that the first part of the ciphertext corresponds to DHIES KEM, but the second part corresponds to a CPA secure DEM. It can be shown that the above scheme is semantically secure under chosen plaintext attacks (i.e. passive attacks) in the random oracle model.

The idea behind our oblivious transfer protocol is for the receiver to create two public keys h_0 and h_1, only one of which it knows the corresponding secret key for. If the receiver knows the secret key for h_b, where b is the bit he is choosing, then he can decrypt for messages encrypted under this key, but not decrypt under the other key. The sender then only needs to encrypt his messages with the two keys. Since the receiver only knows one secret key he can only decrypt one of the message.

To implement this idea concretely, the sender first selects a random element c in G, it is important that the receiver does not know the discrete logarithm of c with respect to g. This value is then sent to the receiver. The receiver then generates two public keys, according to his bit b, via first generating $x \in \mathbb{Z}_q$ and then computing

$$h_b = g^x, \ h_{1-b} = c/h_b.$$

Notice, that the receiver knows the underlying secret key for h_b, but he does not know the secret key for h_{1-b} since he does not know the discrete logarithm of c with respect to g. These two public key values are then sent to the sender. The sender then encrypts message m_0 using the key h_0 and message m_1 using key h_1, i.e. the sender computes

$$\begin{aligned} e_0 &= \left(g^{k_0}, m_0 \oplus H({h_0}^{k_0})\right), \\ e_1 &= \left(g^{k_1}, m_1 \oplus H({h_1}^{k_1})\right), \end{aligned}$$

for two random integers k_0, $k_1 \in \mathbb{Z}_q$. These two ciphertexts are then sent to the receiver who then decrypts the bth one using his secret key x.

From the above description we can obtain some simple optimisations. Firstly, the receiver does not need to send both h_0 and h_1 to the sender, since the sender can always compute h_1 from h_0 by computing c/h_0. Secondly, we can use the same value of $k = k_0 = k_1$ in the two encryptions. We thus obtain the following oblivious transfer protocol

Sender		Receiver
$c \in G$	$\xrightarrow{c}$	
		$x \in \mathbb{Z}_q$,
		$h_b = g^x$,
	$\xleftarrow{h_0}$	$h_{1-b} = c/h_b$,
$h_1 = c/h_0$		
$k \in \mathbb{Z}_q$		
$c_1 = g^k$		
$e_0 = m_0 \oplus H({h_0}^k)$		
$e_1 = m_1 \oplus H({h_1}^k)$	$\xrightarrow{c_1, e_0, e_1}$	
		$m_b = e_b \oplus H({c_1}^x)$.

So does this respect the two conflicting security requirements of participants? First, note that the sender cannot determine the hidden bit b of the receiver since the value h_0 sent from the receiver is simply a random element in G. Then we note that the receiver can learn nothing about m_{1-b} since to do this they would have to be able to compute the output of H on the value h_{1-b}^k, which would imply contradicting the fact that H acts as a random oracle or being able to solve the Diffie–Hellman problem in the group G.

Chapter Summary

- We introduced the idea of protocols between mutually untrusting parties, and introduced commitment and oblivious transfer as two simple examples of such protocols.
- A commitment scheme allows one party to bind themselves to a value, and then reveal it later.
- A commitment scheme needs to be both binding and concealing. Efficient schemes exist which are either information theoretically binding or information theoretically concealing, but not both.
- An oblivious transfer protocol allows a sender to send one of two messages to a recipient, but he does not know which message is actually obtained. The receiver also learns nothing about the other message which was sent.

Further Reading

The above oblivious transfer protocol originally appeared, in a slightly modified form in the paper by Bellare and Micali. The paper by Naor and Pinkas discusses a number of optimisations of the oblivious transfer protocol which we presented above. In particular it presents mechanisms to perform efficiently 1-out-of-N oblivious transfer.

M. Bellare and S. Micali. *Non-interactive oblivious transfer and applications.* In Advances in Cryptology – Crypto '89, Springer-Verlag LNCS 435, 547–557, 1990.

M. Naor and B. Pinkas. *Efficient oblivious transfer protocols.* In SIAM Symposium on Discrete Algorithms – SODA 2001.

CHAPTER 26

Secure Multi-Party Computation

Chapter Goals

- To introduce the concept of multi-party computation.
- To present a two-party protocol based on Yao's garbled circuit construction.
- To present a multi-party protocol based on Shamir secret sharing.

1. Introduction

Secure multi-party computation is an area of cryptography which deals with two or more players computing a function on their private inputs. They wish to do so in a way which means that their private inputs still remain private. Of course depending on the function being computed, some information about the inputs may leak. The classical example is the so-called millionaires problem; suppose a bunch of millionaires have a lunch time meeting at an expensive restaurant and decide that the richest of them will pay the bill. However, they do not want to reveal their actual wealth to each other. This is an example of a secure multi-party computation. The inputs are the values x_i, which denote the wealth of each party, and the function to be computed is

$$f(x_1, \ldots, x_n) = i \text{ where } x_i > x_j \text{ for all } i \neq j.$$

Clearly if we compute such a function then some information about party i's value leaks, i.e. that it is greater than all the other values. However, we require in secure multi-party computation that this is the only information which leaks.

One can consider a number of our previous protocols as being examples of secure multi-party computation. For example, the voting protocol given previously involves the computation of the result of each party voting, without anyone learning the vote being cast by a particular party.

One solution to securely evaluating a function is for all the parties to send their inputs to a trusted third party. This trusted party then computes the function and passes the output back to the parties. However, we want to remove such a trusted third party entirely. Intuitively a multi-party computation is said to be secure if the information which is leaked is precisely that which would have leaked if the computation had been conducted by encrypting messages to a trusted third party.

This is not the only security issue which needs to be addressed when considering secure multi-party computation. There are two basic security models: In the first model the parties are guaranteed to follow the protocols, but are interested in breaking the privacy of their fellow participants. Such adversaries are called honest-but-curious, and they in some sense correspond to passive adversaries in other areas of cryptography. Whilst honest-but-curious adversaries follow the protocol, a number of them could combine their different internal data so as to subvert the security of the non-corrupt parties. In the second model the adversaries can deviate from the protocol and may wish to pass incorrect data around so as to subvert the computation of the function. Again we allow such adversaries to talk to each other in a coalition. Such adversaries are called malicious.

There is a problem though, if we assume that communication is asynchronous, which is the most practically relevant situation, then some party must go last. In such a situation one party may have learnt the outcome of the computation, but one party may not have the value yet (namely the party which receives the last message). Any malicious party can clearly subvert the protocol by not sending the last message. Usually malicious adversaries are assumed not to perform such an attack. A protocol which is said to be secure against an adversary which can delete the final message is said to be fair.

In what follows we shall mainly explain the basic ideas behind secure multi-party computation in the case of honest-but-curious adversaries. We shall touch on the case of malicious adversaries for one of our examples though, as it provides a nice example of an application of various properties of Shamir secret sharing.

If we let n denote the number of parties which engage in the protocol, we would like to create protocols for secure multi-party computation which are able to tolerate a large number of corrupt parties. It turns out that there is a theoretical limit as to the number of parties which can be tolerated as being corrupt. For the case of honest-but-curious adversaries we can tolerate up to $n/2$ corrupt parties. However, for malicious adversaries we can tolerate up to $n/2$ corrupt parties only if we base our security on some computational assumption, for example the inability to break a symmetric encryption scheme. If we are interested in perfect security then we can only tolerate up to $n/3$ corrupt parties in the malicious case.

Protocols for secure multi-party computation usually fall into one of two distinct families. The first is based on an idea of Yao called a garbled circuit or Yao circuit, in this case one presents the function to be computed as a binary circuit, and then one "encrypts" the gates of this circuit to form the garbled circuit. This approach is clearly based on a computational assumption, i.e. that the encryption scheme is secure. The second approach is based on secret sharing schemes; here one usually represents the function to be computed as an arithmetic circuit. In this second approach one uses a perfect secret sharing scheme to obtain perfect security.

It turns out that the first approach seems better suited to the case where there are two parties, whilst the second approach is better suited to the case of three of more parties. In our discussion below we will present a computationally secure solution for the two party case in the presence of honest-but-curious adversaries, based on Yao circuits. This approach can be extended to more than two parties, and to malicious adversaries, but doing this is beyond the scope of this book. We then present a protocol for the multi-party case which is perfectly secure. We sketch two versions, one which provides security against honest-but-curious adversaries and one which provides security against malicious adversaries.

2. The Two-Party Case

We shall in this section consider the method of secure multi-party computation based on garbled circuits. We suppose there are two parties A and B each of whome have input x and y, and that A wishes to compute $f_A(x, y)$ and B wishes to compute $f_B(x, y)$. Recall this needs to be done without B learning anything about x or $f_A(x, y)$, except from what he can deduce from $f_B(x, y)$ and y, with a similar privacy statement applying to A's input and outputs.

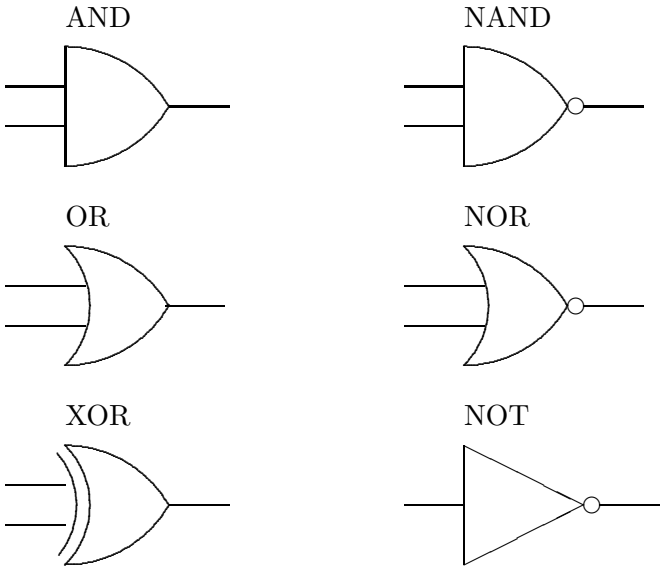
First note that it is enough for B to receive the output of a related function f. To see this we let A have an extra secret input k which is as long as the maximum output of her function $f_A(x, y)$. If we can create a protocol in which B learns the value of the function

$$f(x, y, k) = (k \oplus f_A(x, y), f_B(x, y)),$$

then B simply sends the value of $k \oplus f_A(x, y)$ back to A who can then decrypt it using k, and so determine $f_A(x, y)$. Hence, we will assume that there is only one function which needs to be computed and that its output will be determined by B.

So suppose $f(x, y)$ is the function which is to be computed, we will assume that $f(x, y)$ is a function which can be computed in polynomial time. There is therefore also a polynomial sized binary circuit which will also compute the output of the function. In the forthcoming example we will be writing out such a circuit, and so in Figure 1, we recall the standard symbols for a binary circuit.

FIGURE 1. The Basic Logic Gates



A binary circuit can be represented by a collection of wires $W = \{w_1, \ldots, w_n\}$ and a collection of gates $G = \{g_1, \ldots, g_m\}$. Each gate is function which takes as input the values of two wires, and produces the value of the output wire. For example suppose g_1 is an AND gate which takes as input wire w_1 and w_2 and produces the output wire w_3. Then gate g_1 can be represented by the following truth table.

w_1	w_2	w_3
0	0	0
0	1	0
1	0	0
1	1	1

In other words the gate g_i represents a function such that

$$0 = g_i(0,0) = g_i(1,0) = g_i(0,1) \text{ and } 1 = g_i(1,1).$$

In Yao's garbled circuit construction for secure multi-party computation a circuit is encrypted as follows:

- For each wire w_i two random cryptographic keys are selected k_i^0 and k_i^1. The first one represents the encryption of the zero value, and the second represents the encryption of the one value.
- For each wire a random value $\rho_i \in \{0, 1\}$ is chosen. This is used to also encrypt the actual wire value. If the actual wire value is v_i then the encrypted, or "external" value, is given by $\mathbb{C}e_i = v_i \oplus \rho_i$.
- For each gate we compute a "garbled table" representing the function of the gate on these encrypted values. Suppose g_i is a gate with input wires w_{i_0} and w_{i_1} and output wire w_{i_2},

then the garbled table is the following four values, for some encryption function E,

$$c_{a,b}^{w_{i_2}} = E_{k_{w_{i_0}}^{a\oplus\rho_{i_0}}, k_{w_{i_1}}^{b\oplus\rho_{i_1}}}\left(k_{w_{i_2}}^{o_{a,b}} \| o_{a,b} \oplus \rho_{i_2}\right) \text{ for } a, b \in \{0,1\}.$$

where $o_{a,b} = g_i(a \oplus \rho_{i_0}, b \oplus \rho_{i_1})$.

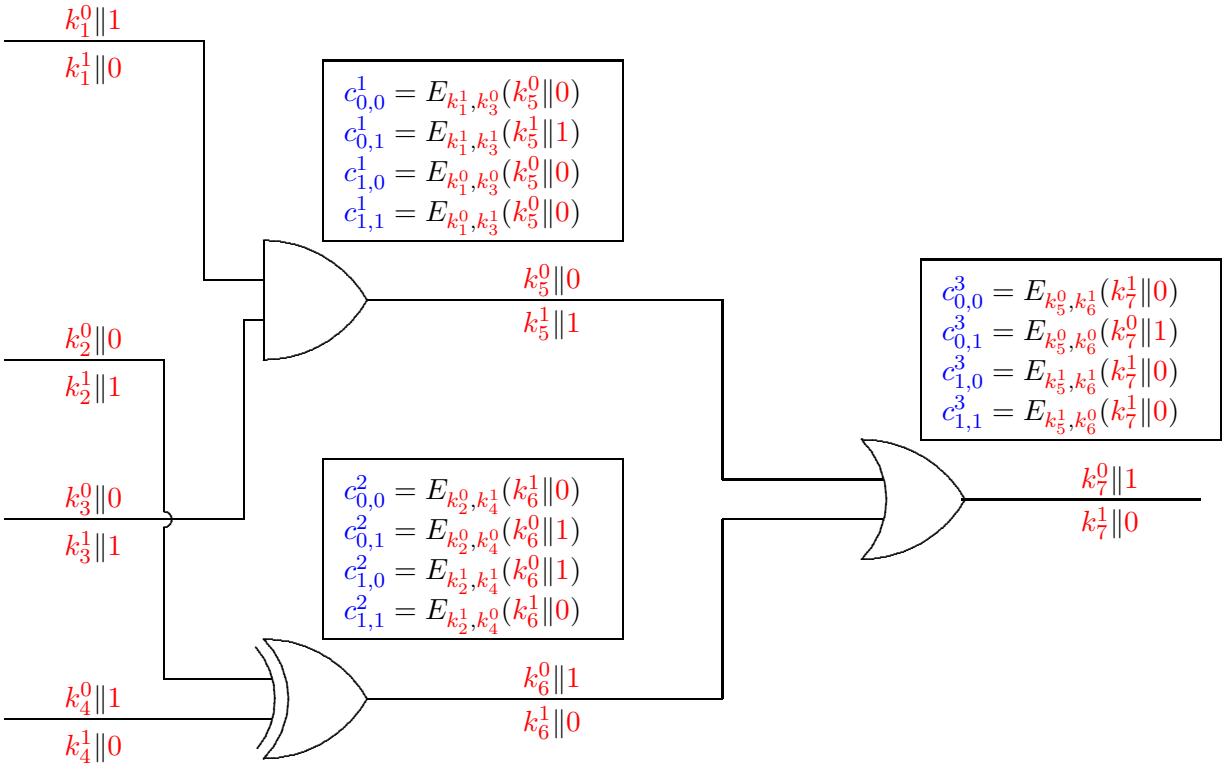
We do not consider exactly what encryption function is chosen, such a discussion is slightly beyond the scope of this book. If you want further details then look in the references at the end of this chapter, or just assume we take an encryption scheme which is suitably secure.

The above may seem rather confusing so we illustrate the method for constructing the garbled circuit with an example. Suppose A and B each have as input two bits, we shall denote A input wires by w_1 and w_2, whilst B's input wires we shall denote by w_3 and w_4. Suppose they now wish to engage in a secure multi-party computation so that B learns the value of the function

$$f(\{w_1, w_2\}, \{w_3, w_4\}) = (w_1 \wedge w_3) \vee (w_2 \oplus w_4).$$

A circuit to represent this function is given in Figure 2.

FIGURE 2. The Garbled Circuit



In Figure 2 we present the garbled values of each wire and the corresponding garbled tables representing each gate. In this example we have the following values of ρ_i,

$$\rho_1 = \rho_4 = \rho_6 = \rho_7 = 1 \text{ and } \rho_2 = \rho_3 = \rho_5 = 0.$$

Consider the first wire, the two garbled values of the wire are k_1^0 and k_1^1, which represent the 0 and 1 value respectively. Since $\rho_1 = 1$ then the external value of the internal 0 value is 1 and the external value of the internal 1 value is 0. Thus we represent the value garbled value of the wire by the pair of pairs

$$(k_1^0\|1, k_1^1\|0).$$

Now we look at the gates, and in particular consider the first gate. The first gate is an AND gate which takes as input the first and third wires. The first entry in this table corresponds to $a = b = 0$. Now the ρ values for the first and third wires are 1 and 0 respectively. Hence, the first entry in the table corresponds to what should happen if the keys k_1^1 and k_3^0 are seen; since

$$1 = 1 \oplus 0 = \rho_1 \oplus a \text{ and } 0 = 0 \oplus 0 = \rho_3 \oplus b.$$

Now the AND gate should produce the 0 output on input of 1 and 0, thus the thing which is encrypted in the first line is the key representing the zero value of the fifth wire, i.e. k_5^0, plus the "external value" of 0, namely $0 = 0 \oplus 0 = 0 \oplus \rho_5$.

We are now in a position to describe Yao's protocol. The protocol proceeds in five phases as follows:

(1) Party A generates the garbled circuit as above, and transmits to party B only the values $c_{a,b}^i$.
(2) Party A then transmits to party B the garbled values of the component of its input wires. For example, suppose in our example that party A's input is $w_1 = 0$ and $w_2 = 0$. Then party A transmits to party B the two values
$$k_1^0 \| 1 \text{ and } k_2^0 \| 0.$$
Note that party B cannot learn the actual values of w_1 and w_2 from these values since it does not know ρ_1 and ρ_2, and the keys k_1^0 and k_2^0 just look like random keys.
(3) Party A and B then engage in an oblivious transfer protocol as in Section 3, for each of party B's input wires. In our example suppose that party B's input is $w_3 = 1$ and $w_4 = 0$. The two parties execute two oblivious transfer protocols, one with A's input
$$k_3^0 \| 0 \text{ and } k_3^1 \| 1,$$
and B's input 1, and one with A's input
$$k_4^0 \| 1 \text{ and } k_4^1 \| 0,$$
and B's input 0. At the end of this oblivious transfer phase party B has learnt
$$k_3^1 \| 1 \text{ and } k_4^0 \| 1.$$
(4) Party A then transmits to party B the values of ρ_i for all of the output wires. In our example he reveals the value of $\rho_7 = 1$.
(5) Finally party B evaluates the circuit using the garbled input wire values he has been given.

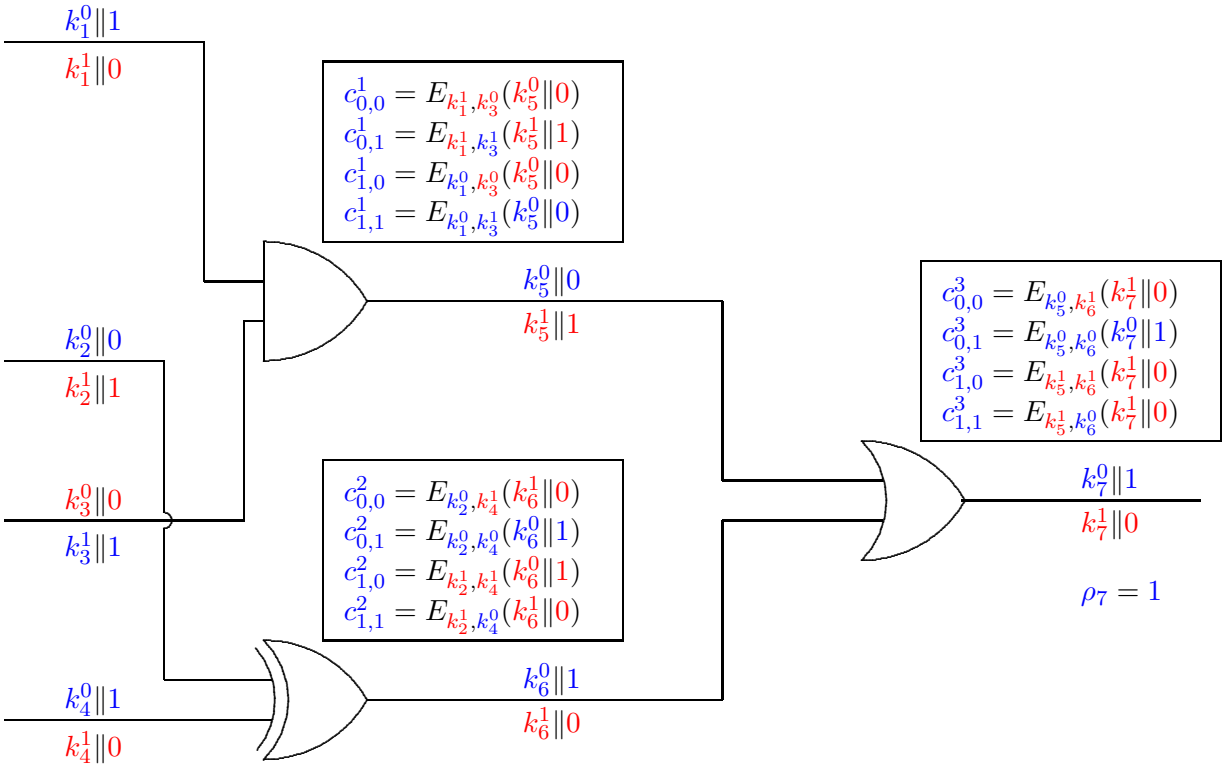
In summary, in the first stage party B only knows the garbled circuit as in the blue items in Figure 2, but by the last stage he knows the blue items in Figure 3.

We now describe how the circuit is evaluated in detail. Please refer to Figure 3 for a graphical description of this. Firstly party B evaluates the AND gate, he knows that the external value of wire one is 1 and the external value of wire three is 1. Thus he looks up the entry $c_{1,1}^1$ in the table and decrypts it using the two keys he knows, i.e. k_1^0 and k_3^1. He then obtains the values $k_5^0 \| 0$. He has no idea whether this represents the zero or one value of the fifth wire, since he has no idea as to the value of ρ_5.

Party B then performs the same operation with the XOR gate. This has input wire 2 and wire 4, for which party B knows that the external values are 0 and 1 respectively. Thus party B decrypts the entry $c_{0,1}^2$ to obtain $k_6^0 \| 1$.

A similar procedure is then carried out with the final OR gate, using the keys and external values of the fifth and sixth wires. This results in a decryption which reveals the value $k_7^0 \| 1$. So the external value of the seventh wire is equal to 1, but party B has been told that $\rho_7 = 1$, and hence the internal value of wire seven will be $0 = 1 \oplus 1$. Hence, the output of the function is the bit 0.

FIGURE 3. Evaluating The Garbled Circuit



So what has Party B learnt from the secure multi-party computation? Party B knows that the output of the final OR gate is zero, which means that the inputs must also be zero, which means that the output of the AND gate is zero and the output of the XOR gate is zero. However, party B knows that the output of the AND gate will be zero, since its own input was zero. However, party B has learnt that party A's second input wire represented zero, since otherwise the XOR gate would not have output zero. So whilst party A's first input remains private, the second input does not. This is what we meant by a protocol keeping the inputs private, bar what could be deduced from the output of the function.

3. The Multi-Party Case: Honest-but-Curious Adversaries

The multi-party case is based on using a secret sharing scheme to evaluate an arithmetic circuit. An arithmetic circuit consists of a finite field $\mathbb{F}_q$, and a polynomial function (which could have many inputs and outputs) defined over the finite field. The idea being that such a function can be evaluated by executing a number of addition and multiplication gates over the finite field.

Given an arithmetic circuit it is clear one could express it as a binary circuit, by simply expanding the addition and multiplication gates of the arithmetic circuit out as their binary circuit equivalents. One can also represent every binary circuit as an arithmetic circuit, since every gate in the binary circuit can be represented as a linear function of the input values to the gate, and their products. Whilst the two representations are equivalent it is clear that some functions are easier to represent as binary circuits and some are easy to represent as arithmetic circuits.

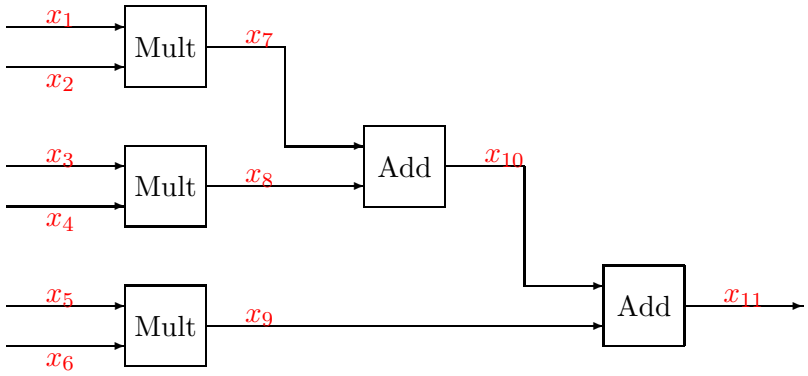
As before we shall present the protocol via a running example. We shall suppose we have six parties, $P_1, \ldots, P_6$ who have six secret values $x_1, \ldots, x_6$ each of which lie in $\mathbb{F}_p$, for some reasonably large prime p. For example we could take $p \approx 2^{128}$, but in our example to make things easier

to represent we will take $p = 101$. The parties are assumed to want to compute the value of the function

$$f(x_1, \ldots, x_6) = x_1 \cdot x_2 + x_3 \cdot x_4 + x_5 \cdot x_6 \pmod{p}.$$

Hence, the arithmetic circuit for this function consists of three multiplication gates and two addition gates, as in Figure 4, where we label the intermediate values as numbered "wires".

FIGURE 4. Graphical Representation of the Example Arithmetic Circuit



The basic protocol idea is as follows, where we use Shamir secret sharing as our secret sharing scheme. The value of each wire x_i is shared between all players, with each player j obtaining a share ${x_i}^{(j)}$. Clearly, if enough players come together then they can determine the value of the wire x_i, by the properties of the secret sharing scheme, and each player can deal shares of his or her own values at the start of the protocol.

The main problem is how to obtain the shares of the outputs of the gates, given shares of the inputs of the gates. Recall in Shamir secret sharing the shared value is given by the constant term of a polynomial f of degree t, with the sharings being the evaluation of the polynomial at given positions corresponding to each participant $f(i)$.

First we consider how to compute the `Add` gates. Suppose we have two secrets a and b which are shared using the polynomials

$$f(X) = a + f_1 X + \cdots + f_t X^t,$$
$$g(X) = b + g_1 X + \cdots + g_t X^t.$$

Each of our parties has a share $a^{(i)} = f(i)$ and $b^{(i)} = g(i)$. Now consider the polynomial

$$h(X) = f(X) + g(X).$$

This polynomial provides a sharing of the sum $c = a + b$, and we have

$$c^{(i)} = h(i) = f(i) + g(i) = a^{(i)} + b^{(i)}.$$

Hence, the parties can compute a sharing of the output of an `Add` gate without any form of communication between them.

Computing the output of a `Mult` gate is more complicated. First we recap on the following property of Lagrange interpolation. If $f(X)$ is a polynomial and we distribute the values $f(j)$ then there is a vector, called the recombination vector, $(r_1, \ldots, r_n)$ such that

$$f(0) = \sum_{i=1}^{n} r_i \cdot f(i).$$

And the same vector works for all polynomials $f(X)$ of degree at most $n-1$.

To compute the `Mult` gate we perform the following four steps. We assume as input that each party has a share of a and b via $a^{(i)} = f(i)$ and $b^{(i)} = g(i)$, where $f(0) = a$ and $g(0) = b$. We wish to compute a sharing $c^{(i)} = h(i)$ such that $h(0) = c = a \cdot b$.

- Each party locally computes $d^{(i)} = a^{(i)} \cdot b^{(i)}$.
- Each party produces a polynomial $\delta_i(X)$ of degree at most t such that $\delta_i(0) = d^{(i)}$.
- Each party i distributes to party j the value $d_{i,j} = \delta_i(j)$.
- Each party j computes $c^{(j)} = \sum_{i=1}^{n} r_i \cdot d_{i,j}$.

So why does this work? Consider the first step, here we are actually effectively computing a polynomial $h'(X)$ of degree at most $2 \cdot t$, with $d^{(i)} = h'(i)$, and $c = h'(0)$. Hence, the only problem with the sharing in the first step is that the underlying polynomial has too high a degree. The main thing to note, is that if

$$2 \cdot t \leq n - 1 \tag{16}$$

then we have $c = \sum_{i=1}^{n} r_i \cdot d^{(i)}$.

Now consider the polynomials $\delta_i(X)$ generated in the second step, and consider what happens when we recombine them using the recombination vector, i.e. set

$$h(X) = \sum_{i=1}^{n} r_i \cdot \delta_i(X).$$

Since the $\delta_i(X)$ are all of degree at most t, the polynomial $h(X)$ is also of degree at most t. We also have that

$$h(0) = \sum_{i=1}^{n} r_i \cdot \delta_i(0) = \sum_{i=1}^{n} r_i \cdot d^{(i)} = c,$$

assuming $2 \cdot t \leq n - 1$. Thus $h(X)$ is a polynomial which *could* be used to share the value of the product. Not only that, but it *is* the polynomial underlying the sharing produced in the final step. To see this notice that

$$h(j) = \sum_{i=1}^{n} r_i \cdot \delta_i(j) = \sum_{i=1}^{n} r_i \cdot d_{i,j} = c^{(j)}.$$

So assuming $t < n/2$ we can produce a protocol which evaluates the arithmetic circuit correctly. We illustrate the method by examining what would happen for our example circuit in Figure 4, with $p = 101$. Recall there are six parties; we shall assume that their inputs are given by

$$x_1 = 20,\ x_2 = 40,\ x_3 = 21,\ x_4 = 31,\ x_5 = 1,\ x_6 = 71.$$

Each party first computes a sharing ${x_i}^{(j)}$ of their secret amongst the six parties. They do this by each choosing a random polynomial of degree $t = 2$ and evaluating it at $j = 1, 2, 4, 5, 6$. The values obtained are then distributed, securely to each party. Hence, each party obtains its row of the following table

	i					
j	1	2	3	4	5	6
1	44	2	96	23	86	83
2	26	0	63	13	52	79
3	4	22	75	62	84	40
4	93	48	98	41	79	10
5	28	35	22	90	37	65
6	64	58	53	49	46	44

As an exercise you should work out the associated polynomials corresponding to each column.

The parties then engage in the multiplication protocol so as to compute sharings of $x_7 = x_1 \cdot x_2$. They first compute their local multiplication, by each multiplying the first two elements in their row of the above table, then they for a sharing of this local multiplication. These sharings of six numbers between six parties are then distributed securely as before. In our example run each party, for this multiplication, obtains the sharings given by the column of the following table

	j					
i	1	2	3	4	5	6
1	92	54	20	91	65	43
2	10	46	7	95	7	46
3	64	100	96	52	69	46
4	23	38	41	32	11	79
5	47	97	77	88	29	1
6	95	34	11	26	79	69

Each party then takes the six values obtained and recovers their share of the value of x_7. We find that the six shares of x_7 are given by

$$x_7{}^{(1)} = 9,\ x_7{}^{(2)} = 97,\ x_7{}^{(3)} = 54,\ x_7{}^{(4)} = 82,\ x_7{}^{(5)} = 80,\ x_7{}^{(6)} = 48.$$

Repeating the multiplication protocol twice more we also obtain a sharing of x_8 as

$$x_8{}^{(1)} = 26,\ x_8{}^{(2)} = 91,\ x_8{}^{(3)} = 38,\ x_8{}^{(4)} = 69,\ x_8{}^{(5)} = 83,\ x_8{}^{(6)} = 80,$$

and x_9 as

$$x_9{}^{(1)} = 57,\ x_9{}^{(2)} = 77,\ x_9{}^{(3)} = 30,\ x_9{}^{(4)} = 17,\ x_9{}^{(5)} = 38,\ x_9{}^{(6)} = 93,$$

We are then left with the two addition gates to produce the sharings of the wires x_{10} and x_{11}. These are obtained by locally adding together the various shared values so that

$$x_{11}{}^{(1)} = x_7{}^{(1)} + x_8{}^{(1)} + x_9{}^{(1)} \pmod{101} = 9 + 26 + 57 \pmod{101} = 92,$$

etc, to obtain

$$x_{11}{}^{(1)} = 92,\ x_{11}{}^{(2)} = 63,\ x_{11}{}^{(3)} = 21,\ x_{11}{}^{(4)} = 67,\ x_{11}{}^{(5)} = 100,\ x_{11}{}^{(6)} = 19.$$

The parties then make public these shares, and recover the hidden polynomial, of degree $t = 2$, which produces these sharings, namely

$$7 + 41X^2 + 44X^2.$$

Hence, the result of the multi-party computation is the value 7.

Now assume that more than t parties are corrupt, in the sense that they collude to try and break the privacy of the non-corrupted parties. The corrupt parties can now come together and recover any of the underlying secrets in the scheme, since we have used Shamir secret sharing using polynomials of degree at most t. It can be shown that, using the perfect secrecy of the Shamir secret sharing scheme that as long as less than or equal to t parties are corrupt then the above protocol is perfectly securely.

However, it is only perfectly secure assuming all parties follow the protocol, i.e. we are in the honest-but-curious model. As soon as we allow parties to deviate from the protocol then they can force the honest parties to produce invalid results. To see this just notice that a dishonest party could simply produce an invalid sharing of its product in the second part of the multiplication protocol above.

4. The Multi-Party Case: Malicious Adversaries

To produce a scheme which is secure against such active adversaries we need to force all parties to either follow the protocol, or we should be able to recover from errors which malicious parties introduce introduce into the protocol. It is the second of these two approaches which we shall follow in this section, by using the error correction properties of the Shamir secret sharing scheme.

As already remarked the above protocol is not secure against malicious adversaries, due to the ability of an attacker to make the multiplication protocol output an invalid answer. To make the above protocol secure against malicious adversaries we make use of various properties of the Shamir secret sharing scheme.

The protocol runs in two stages: The preprocessing stage does not involve any of the secret inputs of the parties, it purely depends on the number of multiplication gates in the circuit. In the main phase of the protocol the circuit is evaluated as in the previous section, but using a slightly different multiplication protocol. Malicious parties can force the preprocessing stage to fail, however if it completes then the honest parties will be able to evaluate the circuit as required.

The preprocessing phase runs as follows. First using the techniques from Chapter 23 a pseudorandom secret sharing scheme, PRSS, and a pseudorandom zero sharing scheme, PRZS, are set up. Then for each multiplication gate in the circuit we compute a random triple of sharings $a^{(i)}$, $b^{(i)}$ and $c^{(i)}$ such that $c = a \cdot b$. This is done as follows:

- Using PRSS generate two random sharings, $a^{(i)}$ and $b^{(i)}$, of degree t.
- Using PRSS generate another random sharing $r^{(i)}$ of degree t.
- Using PRZS generate a sharing $z^{(i)}$, of degree $2 \cdot t$ of zero.
- Each player then locally computes
$$s^{(i)} = a^{(i)} \cdot b^{(i)} - r^{(i)} + z^{(i)}.$$
Note this local computation will produce a degree $2 \cdot t$ sharing of the value $s = a \cdot b - r$.
- Then the players broadcast their values $s^{(i)}$ and try to recover s. This can be done always using the error correction properties of Reed–Solomon codes assuming the number of malicious parties is bounded by $n/3$.
- Now the players locally compute the shares $c^{(i)}$ from $c^{(i)} = s + r^{(i)}$.

Assuming the above preprocessing phase completes successfully all we need do is specify how the parties implement a `Mult` in the presence of malicious adversaries. We assume the inputs to the multiplication gate are given by $x^{(i)}$ and $y^{(i)}$ and we wish to compute a sharing $z^{(i)}$ of the produce $z = x \cdot y$. From the preprocessing stage, the parties also have for each gate, a triple of shares $a^{(i)}$, $b^{(i)}$ and $c^{(i)}$ such that $c = a \cdot b$. The protocol for the multiplication is then as follows:

- Compute locally, and then broadcast, the values $d^{(i)} = x^{(i)} - a^{(i)}$ and $e^{(i)} = y^{(i)} - b^{(i)}$
- Reconstruct the values of $d = x - a$ and $d = y - b$.
- Locally compute the shares
$$z^{(i)} = d \cdot e + d \cdot b^{(i)} + e \cdot a^{(i)} + c^{(i)}.$$

Note that the reconstruction in the second step can be completed as long as there are at most $t < n/3$ malicious parties. The computation in the last step recovers the valid shares due to the linear nature of the Shamir secret sharing scheme and the underlying equation

$$\begin{aligned} d \cdot e + d \cdot b + e \cdot a + c &= (x - a) \cdot (y - b) + (x - a) \cdot b + (y - b) \cdot a + c \\ &= ((x - a) + a) \cdot ((y - b) + b) \\ &= x \cdot y = z. \end{aligned}$$

Chapter Summary

- We have explained how to perform two party secure computation in the precense of honest-but-curious adversaries using Yao's garbled circuit construction.
- For the many party case we have presented a protocol based on evaluating arithmetic, as opposed to binary, circuits which is based on Shamir secret sharing.
- The main issue with this latter protocol is how to evaluate the multiplication gates. We presented two methods: The first, simpler, method is applicable when one is only dealing with honest-but-curious adversaries, the second, more involved, method is for the case of malicious adversaries.

Further Reading

The original presentation of Yao's idea appears in FOCS 1986, It can be transformed into a scheme for malicious adversaries using a general technique of Goldreich et. al. The discussion of the secret sharing based solution for the honest and malicious cases closely follows the treatment in Damgård et. al.

I. Damgård, M. Geisler, M. Kroigaard and J.B. Nielsen. *Asynchronous multiparty computation: Theory and implementation* IACR e-print `eprint.iacr.org/2008/415`.

O. Goldreich, S. Micali and A. Wigderson. *How to play any mental game.* In Symposium on Theory of Computing – STOC 1987, ACM, 218–229, 1987.

A.C. Yao. *How to generate and exchange secrets.* In Foundations of Computer Science – FOCS 1986, IEEE, 162–167, 1986.