5.1 The Basic Protocol

Protocol 5.1 (oblivious transfer from errorless homomorphic encryption):

- **Inputs:** *The sender S has a pair of strings (x_0, x_1) for input; the receiver R has a bit σ. Both parties have the security parameter 1^n as auxiliary input. (In order to satisfy the constraints that all inputs are of the same length, it is possible to define $|x_0| = |x_1| = k$ and give the receiver $(\sigma, 1^{2k-1})$.)*

- **Assumption:** *We assume that the group determined by the homomorphic encryption scheme with security parameter n is large enough to contain all strings of length k. Thus, if the homomorphic encryption scheme only works for single bits, we will only consider $k = 1$ (i.e., bit oblivious transfer).*

- **The protocol:**

  1. *The receiver R chooses two sets of two pairs of keys:*
     (a) $(pk_1^0, sk_1^0), (pk_2^0, sk_2^0) \leftarrow G(1^n)$ *using random coins* r_G^0, *and*
     (b) $(pk_1^1, sk_1^1), (pk_2^1, sk_2^1) \leftarrow G(1^n)$ *using random coins* r_G^1

     R sends (pk_1^0, pk_2^0) and (pk_1^1, pk_2^1) to the sender S.
  2. Key-generation challenge:
     (a) *S chooses a random coin $b \in_R \{0,1\}$ and sends b to R.*
     (b) *R sends S the random-coins r_G^b that it used to generate (pk_1^b, pk_2^b).*
     (c) *S checks that the public keys output by the key-generation algorithm G when given input 1^n and the appropriate portions of the random-tape r_G^b equal pk_1^b and pk_2^b. If this does not hold, or if R did not send any message here, S outputs $\mathsf{corrupted}_R$ and halts. Otherwise, it proceeds.*
     Denote $pk_1 = pk_1^{1-b}$ and $pk_2 = pk_2^{1-b}$.
  3. *R chooses two random bits $\alpha, \beta \in_R \{0,1\}$. Then:*
     (a) *R computes*

     $$c_0^1 = E_{pk_1}(\alpha) \qquad c_0^2 = E_{pk_2}(1-\alpha)$$
     $$c_1^1 = E_{pk_1}(\beta) \qquad c_1^2 = E_{pk_2}(1-\beta)$$

     using random coins r_0^1, r_0^2, r_1^1 and r_1^2, respectively.
     (b) *R sends (c_0^1, c_0^2) and (c_1^1, c_1^2) to S.*
  4. Encryption-generation challenge:
     (a) *S chooses a random bit $b' \in_R \{0,1\}$ and sends b' to R.*
     (b) *R sends $r_{b'}^1$ and $r_{b'}^2$ to S (i.e., R sends an opening to the ciphertexts $c_{b'}^1$ and $c_{b'}^2$).*
     (c) *S checks that one of the ciphertexts $\{c_{b'}^1, c_{b'}^2\}$ is an encryption of 0 and the other is an encryption of 1. If not (including the case that no message is sent by R), S outputs $\mathsf{corrupted}_R$ and halts. Otherwise, it continues to the next step.*
  5. *R sends a "re-ordering" of the ciphertexts $\{c_{1-b'}^1, c_{1-b'}^2\}$. Specifically, if $\sigma = 0$ then it sets c_0 to be the ciphertext that is an encryption of 1, and sets c_1 to be the ciphertext that is an encryption of 0. Otherwise, if $\sigma = 1$ then it sets c_0 to be the encryption of 0, and c_1 to be the encryption of 1. (Only the ordering needs to be sent and not the actual ciphertexts. Furthermore, this can be sent together with the openings in Step 4b.)*

6. *S uses the homomorphic property and c_0, c_1 as follows.*
   (a) *S computes $\tilde{c}_0 = x_0 \cdot_E c_0$ (this operation is relative to the key pk_1 or pk_2 depending if c_0 is an encryption under pk_1 or pk_2)*
   (b) *S computes $\tilde{c}_1 = x_1 \cdot_E c_1$ (this operation is relative to the key pk_1 or pk_2 depending if c_1 is an encryption under pk_1 or pk_2)*

   S sends $\tilde{c}_0$ and $\tilde{c}_1$ to R. (Notice that one of the ciphertexts is encrypted with key pk_1 and the other is encrypted with key pk_2.)
7. *If $\sigma = 0$, the receiver R decrypts $\tilde{c}_0$ and outputs the result (if $\tilde{c}_0$ is encrypted under pk_1 then R outputs $x_0 = D_{sk_1}(\tilde{c}_0)$; otherwise it outputs $x_0 = D_{sk_2}(\tilde{c}_0)$). Otherwise, if $\sigma = 1$, R decrypts $\tilde{c}_1$ and outputs the result.*
8. *If at any stage during the protocol, S does not receive the next message that it expects to receive from R or the message it receives is invalid and cannot be processed, it outputs* abort_R *(unless it was already instructed to output* $\mathsf{corrupted}_R$*). Likewise, if R does not receive the next message that it expects to receive from S or it receives an invalid message, it outputs* abort_S*.*

We remark that the reordering message of Step 5 can actually be sent by R together with the message in Step 4b. Furthermore, the messages of the key-generation challenge can be piggybacked with later messages, as long as they conclude before the final step. We therefore have that the number of rounds of communication can be exactly *four* (each party sends two messages).

Before proceeding to the proof of security, we present the intuitive argument showing why Protocol 5.1 is secure. We begin with the case that the receiver is corrupt. First note that if the receiver follows the instructions of the protocol, it learns only a single value x_0 or x_1. This is because one of c_0 and c_1 is an encryption of 0. If it is c_0, then $\tilde{c}_0 = x_0 \cdot_E c_0 = E_{pk}(0 \cdot x_0) = E_{pk}(0)$ (where $pk \in \{pk_1, pk_2\}$, and so nothing is learned about x_0; similarly if it is c_1 then $\tilde{c}_1 = E_{pk}(0)$ and so nothing is learned about x_1. However, in general, the receiver may not generate the encryptions $c_0^1, c_1^1, c_0^2, c_1^2$ properly (and so it may that at least one of the pairs (c_0^1, c_0^2) and (c_1^1, c_1^2) are *both* encryptions of 1, in which case the receiver could learn both x_0 and x_1). This is prevented by the encryption-generation challenge. That is, if the receiver tries to cheat in this way then it is guaranteed to be caught with probability at least 1/2. The above explains why a malicious receiver can learn only one of the outputs, unless it is willing to be caught cheating with probability 1/2. This therefore demonstrates that "privacy" holds. However, we actually need to prove security via simulation, which involves showing how to *extract* the receiver's implicit input and how to *simulate* its view. Extraction works by first providing the corrupted receiver with the encryption-challenge bit $b' = 0$ and then rewinding it and providing it with the challenge $b' = 1$. If the corrupted receiver replies to both challenges, then the simulator can construct σ from the opened ciphertexts and the reordering provided. Given this input, the simulation can be completed in a straightforward manner; see the proof below. A crucial point here is that if the receiver does not reply to both challenges then an honest sender would output $\mathsf{corrupted}_R$ with probability 1/2, and so this corresponds to a cheat_R input in the ideal world.

We now proceed to discuss why the protocol is secure in the presence of a corrupt sender. In this case, it is easy to see that such a sender cannot learn anything about the receiver's input because the encryption scheme is semantically secure (and so a corrupt sender cannot determine σ from the unopened ciphertexts). However, as above, we need to show how extraction and simulation works. Extraction here works by providing encryptions so that in one of the pairs (c_0^1, c_0^2) or (c_1^1, c_1^2) both of the encrypted values are 1. If this pair is the one used (and not the one opened) , then we