ETH Zurich, Department of Computer Science
SS 2020

Prof. Ueli Maurer
Dr. Martin Hirt
Chen-Da Liu Zhang

Cryptographic Protocols
Solution to Exercise 8

8.1 Impossibility and Feasibility Proofs

a) If the players want to compute $b_1 \oplus b_2$, then the next protocol is passively secure:
Party P_1 sends b_1 to P_2, then party P_2 sends b_2 to P_1, and both output $b_1 \oplus b_2$.
Observe that P_1 can obtain the input b_2 from his input b_1 and the output $b_1 \oplus b_2$. Also, P_2 can obtain the input b_1 from his input b_2 and the output $b_1 \oplus b_2$. Hence, regardless of who the adversary corrupts, he does not learn anything beyond what is leaked by the output.

b) If the output is constant (0 or 1), the parties can trivially output the constant without exchanging any messages. We only need to focus on the case where the number of ones is two.
If there are two ones, the outputs can be either the first input $(0, 0, 1, 1)$, the second input $(0, 1, 0, 1)$, the XOR of the inputs $(0, 1, 1, 0)$, or the negation of one of the three cases.
If the output is equal to the input of a party P_i, the protocol consists of one message, where P_i sends his input b_i to the other party, and both output b_i. In the previous subtask we saw a passively secure protocol that computes the XOR.
For the negated functions, it is enough to execute the protocol for the non-negated function, and at the end negate the output.

c) Now we show that if the output vector contains exactly a one, it is impossible to construct a protocol for such a function. We make the argument for the case where the vector is $(1, 0, 0, 0)$, since all cases are similar. In this case, the function is $\neg b_0 \wedge \neg b_1$. We sketch a reduction to the impossibility proof for the AND function presented in the lecture. Assume there is a protocol π that computes the function $\neg b_0 \wedge \neg b_1$. In order to construct a protocol that computes the AND function, the parties negate their inputs, and execute the protocol π on inputs $\neg b_0$ and $\neg b_1$.
Finally, if the vector has three ones, it is enough to execute the protocol for the negated function which has only a one, and negate the output.

8.2 Not Sending Values

Note first that, given at least $t + 1$ of the n shares, a_ℓ can be computed using Lagrange interpolation. That is, let $S \subseteq \{1, \ldots, n\} \setminus \{\ell\}$ with $|S| \geq t + 1$. Then,

$$a_\ell = \sum_{j \in S} w_j a_j \qquad \text{where} \qquad w_j = \prod_{\substack{k \in S \\ k \neq j}} \frac{\alpha_\ell - \alpha_k}{\alpha_j - \alpha_k}.$$

Since the weights w_j are constant, a_ℓ is a linear function of the shares a_j for $j \in S$.

The above leads to the following protocol idea: Each player P_i re-shares his share a_i as $[a_i] = (a_{i1}, \ldots, a_{in})$ among the players. This is followed by an accusation phase to make sure that all honest players agree on the same set S of players who have performed the re-sharing. Then, the players compute a sharing of a_ℓ (using only local operations on their respective shares) as shown above and reconstruct the value.

The actual protocol:

1. SHARING: Every player P_i shares his share a_i among all players. Let a_{ij} denote P_j's share of a_i.
2. ACCUSATIONS: If a player P_j does not receive his share a_{ij} from some player P_i, then P_j complains about this by broadcasting an accusation. As an answer, P_i must broadcast the value a_{ij}. If P_i does not do so, he is disqualified. Denote by S the set of players that have *not* been disqualified.
3. COMPUTING THE SHARE OF a_ℓ: Every player P_i (locally) computes his share $a_{\ell i}$ of the value a_ℓ as follows:

$$a_{\ell i} = \sum_{j \in S} w_j a_{ji} \qquad \text{where} \qquad w_j = \prod_{\substack{k \in S \\ k \neq j}} \frac{\alpha_\ell - \alpha_k}{\alpha_j - \alpha_k}.$$

4. RECONSTRUCTING a_ℓ: Every P_i sends his share $a_{\ell i}$ to all other players. Let S_i denote the set of players from which P_i has received a share. P_i computes a_ℓ as follows:

$$a_\ell = \sum_{j \in S_i} w_j a_{\ell j} \qquad \text{where} \qquad w_j = \prod_{\substack{k \in S_i \\ k \neq j}} \frac{\alpha_k}{\alpha_k - \alpha_j}.$$

It is important that all (honest) players choose the same set S (as done in Step 2). Otherwise, they would execute Step 3 on different sharings and reconstruction would not work anymore. It is easy to verify that this protocol does not violate privacy.

8.3 Perfectly Binding/Hiding Commitments

We consider *perfectly correct* commitment schemes with a *non-interactive* COMMIT *phase*. Such a commitment scheme can be characterized by a function $C : \mathcal{X} \times \mathcal{R} \to \mathcal{B}$ that maps a value $x \in \mathcal{X}$ and a randomness string r from some randomness space $\mathcal{R}$ to a blob $b = C(x, r)$ in some blob space $\mathcal{B}$. The OPEN phase simply consists of the prover's sending (x, r) to the verifier, who checks that $C(x, r) = b$.

In the following, denote by $\mathcal{B}_x := \operatorname{im} C(x, \cdot)$ for $x \in \mathcal{X}$.

a) Let $x \neq x'$. Perfectly binding means that $\mathcal{B}_x \cap \mathcal{B}_{x'} = \emptyset$, whereas perfectly hiding means that $C(x, R)$ and $C(x', R)$ are identically distributed random variables for $R \in_R \mathcal{R}$. This requires in particular that $\mathcal{B}_x = \mathcal{B}_{x'}$, which contradicts $\mathcal{B}_x \cap \mathcal{B}_{x'} = \emptyset$.

b) Subtasks **b)** and **c)** are discussed simultaneously in **c)**.

c) Note that in all cases, the combined scheme is a string commitment $C(x, (r_1, r_2))$.

1. HIDING: The computational hiding property of C_B cannot be broken by additionally adding the blob of the perfectly hiding scheme C_H.[1]
   BINDING: As C_B is perfectly binding, this is also true for the combined scheme $(C_H(x, r_1), C_B(x, r_2))$, since $C(x, (r_1, r_2)) = C(x', (r_1', r_2'))$ implies that $C_B(x, r_2) = C_B(x', r_2')$.

[1] Formally, this would have to be proved via a reduction.

2. HIDING: Clearly, the scheme is perfectly hiding as $C_H(C_B(x, r_1), r_2)$ perfectly hides $C_B(x, r_1)$ and thereby x.
   BINDING: Assume for contradiction one could efficiently come up with $x \neq x'$, (r_1, r_2), and (r'_1, r'_2) such that $C(x, (r_1, r_2)) = C(x', (r'_1, r'_2))$. Then, by the fact that C_B is perfectly binding, $y := C_B(x, r_1) \neq C_B(x', r'_1) =: y'$, one can efficiently come up with $y \neq y'$, r_2, and r'_2 such that $C_H(y, r_2) = C_H(y', r'_2)$, which breaks the (computational) binding property of C_H.
3. HIDING: Clearly, the scheme is perfectly hiding as $C_H(x, r_1)$ perfectly hides x.
   BINDING: Assume for contradiction one could efficiently come up with $x \neq x'$, (r_1, r_2), and (r'_1, r'_2) such that $C(x, (r_1, r_2)) = C(x', (r'_1, r'_2))$. Then, by the fact that C_B is perfectly binding, $y := C_H(x, r_1) = C_H(x', r'_1) =: y'$, one can efficiently come up with $x \neq x'$, r_1, and r'_1 such that $C_H(x, r_1) = y = C_H(x', r'_1)$, which breaks the (computational) binding property of C_H.