6.1 Homomorphic Commitments

Note that a blob committing to 0 is a quadratic residue, and, since \( t \) is a quadratic non-residue with \( \left( \frac{t}{m} \right) = +1 \), a blob committing to 1 is a quadratic non-residue \( b \) with \( \left( \frac{b}{m} \right) = +1 \). Thus, the scheme is of type \( B \), where the computational hiding property relies on the QR assumption, which states that modulo an RSA prime \( m \) it is hard to distinguish quadratic residues from quadratic non-residues with \( \left( \frac{b}{m} \right) = +1 \).

a) Denote by \( b_0 = r_0^2 t^{x_0} \) and \( b_1 = r_1^2 t^{x_1} \) two blobs to bits \( x_0 \) and \( x_1 \), respectively. By multiplying \( b_0 \) and \( b_1 \), one obtains
\[
b = b_0 \cdot b_1 = r_0^2 \cdot r_1^2 \cdot t^{x_0 + x_1}.
\]
This is a commitment to \( x_0 \oplus x_1 \): If \( x_0 = x_1 \) (i.e., \( x_0 \oplus x_1 = 0 \)), then \( b \) is a quadratic residue (with square root \( r_0 r_1 \) if \( x_0 = x_1 = 0 \) and \( r_0 r_1 t \) if \( x_0 = x_1 = 1 \)). If \( x_0 \neq x_1 \) (i.e., \( x_0 \oplus x_1 = 1 \)), then \( b \) is a quadratic non-residue with \( \left( \frac{b}{m} \right) = +1 \).

b) Let \( b_0 = r_0^2 t^x \) be the blob to \( x \). By multiplying \( b_0 \) by \( t \) one obtains
\[
b_1 = b_0 \cdot t = r_0^2 \cdot t^{x+1}.
\]
If \( x = 0 \), \( b_1 \) is a quadratic non-residue and thus a commitment to 1. If \( x = 1 \), \( b_1 \) is a quadratic residue and thus a commitment to 0.

c) If all binary operations could be implemented in such a fashion, the BCC protocol would become substantially simpler: At the beginning, Peggy would commit to the satisfying input, and then Vic would evaluate the circuit on the blobs. In the end, Peggy would prove that the resulting blob \( b \) is indeed a commitment to one by proving that \( bt \) is a quadratic residue using the Fiat-Shamir protocol.

d) As shown in a), if \( x_0 = x_1 \), \( b_0 \cdot b_1 \) is a quadratic residue, a fact that Peggy can prove using the Fiat-Shamir protocol. Moreover, if \( x_0 \neq x_1 \), then \( b := b_0 \cdot b_1 \) is a quadratic non-residue with \( \left( \frac{b}{m} \right) = +1 \) and thus \( b_0 \cdot b_1 \cdot t \) is a quadratic residue, which, again, can be proved using the Fiat-Shamir protocol.

6.2 Permuted Truth Tables

a) Peggy chooses a random permuted truth table for the \( \land \)-function and commits to its elements. Vic chooses a random challenge bit \( c \) and sends it to Peggy. If \( c = 0 \), then Peggy opens the whole table and Vic checks if it is a permuted \( \land \)-table. If \( c = 1 \), Peggy takes the blobs \( (d_1, d_2, d_3) \) from the row corresponding to the triple \( (b_1, b_2, b_3) \) and proves (using the ZK protocol for equality) that \( \forall i \in \{1, 2, 3\} \) \( d_i \) and \( c_i \) are commitments of the same value.
Note that the commitments used in the above construction are of type B (i.e., perfectly binding). We show that the above protocol is a zero-knowledge proof of the statement “the committed values \((b_1, b_2, b_3)\) corresponding to the commitments \((c_1, c_2, c_3)\) satisfy the relation \(b_1 \land b_2 = b_3\).”

**Completeness:** Follows immediately from the completeness of the protocol for blob equality.

**Soundness:** Assume that \(b_1 \land b_2 \neq b_3\). If Peggy commits to a valid permuted truth table in the first step, Peggy cannot answer the challenge \(c = 1\) as there is no row in this table with with commitments corresponding to \(b_1, b_2, b_3\). If Peggy commits to an invalid table, then she cannot answer the challenge \(c = 0\), as the commitment is binding. Hence, the cheating probability of Peggy for each round is approximately \(1/2\) (the “approximately” stems from the fact that, in case \(c = 1\), Peggy might still be able, with some small probability, to cheat in the equality proof).

**Zero-Knowledge:** We prove the (computational) zero-knowledge property only informally. We need to show that there exists an efficient simulation \(S\) producing a transcript which is (computationally) indistinguishable from the transcript resulting from a real protocol execution between the prover \(P\) and (a possibly dishonest) verifier \(V'\).

The simulator \(S\) can produce a transcript as follows: First, \(S\) computes a valid permuted truth table and commits to it. If \(V'\) sends the challenge \(c = 0\), the simulator opens the committed table. If \(V'\) sends \(c = 1\), \(S\) uses the simulator \(S'\) for the blob equality protocol to compute a transcript of a proof of equality for \(c_i = d_i\) \((i = 1 \ldots 3)\), where the \(d_i\)'s are commitments corresponding to a randomly chosen row of the permuted truth table. Note that, by the computational hiding property of the commitments, the transcript produced by \(S'\) is computationally indistinguishable from the real interaction even if the \(d_i\)'s are commitments to different values than those in the \(c_i\)'s.

b) If Peggy knows the input to the circuit, then she can compute (by evaluating the circuit in a gate-by-gate manner) the bits on the wires. She commits to all those bits and sends the blobs to Vic. Subsequently, she uses the protocol from a) for each gate (\(\neg\)-gates are treated similarly to \(\land\)-gates) to prove that the committed values are consistent with the circuit. To convince Vic that the output of the circuit is in fact 1, Peggy and Vic use a fixed commitment of 1, i.e., a commitment that is hard-coded into the protocol.

c) In the BCC protocol from the lecture, when processing the circuit, Peggy blinds every wire using a random bit. In the protocol from b), this is not necessary, but we need the additional zero-knowledge proofs of equality of committed values.

### 6.3 Sudoku

In the following we use a commitment scheme of Type B.

a) The following protocol is a possible solution for this task:

**Phase 1:** Peggy chooses a random permutation \(\sigma : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}\). For each cell with value \(v\), she commits to \(\sigma(v)\).

**Phase 2:** Vic randomly chooses one of the following challenges: a row, a column, a subgrid, or preprinted values.

**Phase 3:** If Vic chose as challenge a region, e.g., a row, Peggy opens all commitments of said region, and Vic checks that all values from \(\{1, \ldots, n\}\) occur. In the case of the “preprinted” challenge, Peggy opens all commitments of preprinted values. Vic checks if \(\sigma\) is indeed a permutation on the preprinted values, i.e., he checks if values which are originally (not) equal, are (not) equal in the permuted version.
b) **Completeness:** If Peggy knows the Sudoku solution, she can answer all challenges, so completeness follows directly.

**Proof of Knowledge:** Note first, that if Peggy can answer all challenges after Phase 1, the solution can be computed from these answers. The knowledge extractor form the proof of Theorem 4.1 needs to be adapted slightly to deal with the fact that not for every pair of different challenges one can extract a solution. We omit this here.

c) **Zero-Knowledge:** In the case of a “region challenge,” the opened values are a random permutation of \(\{1, \ldots, n\}\). In the other case the distribution is a (random) injection of the preprinted values into \(\{1, \ldots, n\}\). Hence one could generate a transcript identically distributed to an actual protocol transcript. Therefore the protocol is zero-knowledge.

### 6.4 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 \(C : \mathcal{X} \times \mathcal{R} \rightarrow \mathcal{B}\) 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 \(C(x, r) = b\) (cf. Section 6.1 of the lecture notes).

In the following, denote by \(\mathcal{B}_x := \text{im} C(x, \cdot)\) 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 \(R \in \mathcal{R}\). 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 \(C_H\).

   **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 \(C(x, r_1) = C(x’, r_1’\).

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 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, since, 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 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, since, 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 \(C_H\).

---

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