We present a protocol for our context, which has better detection protocol than the generic construction presented by Aumann and Lindell []. Recall that in our context there are two parties in the protocol: party 1 (client) has input $x$ and party 2 (server) has input $y$. They want to jointly compute the functionality $(x, y) \longmapsto (\delta(H(x) = y), \delta(H(x) = y))$ where $\delta_{H(x)=y}$ is equal to 1 if $H(x) = y$; otherwise it is 0. If client gets output of 1, it means that client was authenticated by the server. In our context, the reader should interpret $x$ as the password and $y$ as the hash of the password (in other words client can only successfully authenticate if they know a password whose hash matches what the server has). The key idea is that the hash function $H$ is included in the functionality, which makes our protocol resilient against weakness suggested in Section **??**. Our protocol description assumes that the reader is familiar with the basics of secure-function evaluation (such as garbled circuit construction and oblivious transfer).

- **(Step 1)** Client creates $l$ garbled circuits $C_1, \cdots, C_l$ of the for $\delta(H(x), y)$. Let server's input $y = y_1 \cdots y_m$ be $m$ bits. The wire keys corresponding to the $j$-bit of server's input for the $i$-th garbled circuit $C_i$ is denoted by $k^0_{i,j}$ and $k^1_{i,j}$. Client sends circuits $C_1, C_2, \cdots, C_l$ to the server.

- **(Step 2)** Client and server execute the $OT^2_1$ protocol $m$ times. In the $j$-th instance of $OT^2_1$ the client acts as a sender with inputs $k^0_{1,j} \| k^0_{2,j} \| \cdots \| k^0_{l,j}$ and $k^1_{1,j} \| k^1_{2,j} \| \cdots \| k^1_{l,j}$ and the server acts the chooser with input $y_j$ (the $j$-th bit of the input). Notice that concatenating the keys prevents the server from learning keys corresponding to different bits, e.g., server cannot learn keys $k^0_{1,j}$ and $k^1_{2,j}$.

- **(Step 3)** Server chooses a random set $S \subseteq \{1, 2, \cdots, l\}$ and sends $S$ to the client.

- **(Step 4)** Client reveals wire keys for circuits $C_j$ such that $j \in S$ to the server (we call this step opening the circuits $C_j$ such that $j \in S$). Client also provides wire keys for its input $x$ for circuits $C_j$, $j \notin S$.

- **(Step 5)** If the circuits $C_j$ ($j \in S$) are not well-formed (the circuits do not compute $\delta(H(x), y)$ or the keys are not consistent with what was sent in step 2), the server sends 0 to the client. Server computes $C_j$ ($j \notin S$) and obtains answers $o_j$ ($j \notin S$). Server sends $\bigwedge_{j \notin S} o_j$ to the client.

It is clear that if both client and server are honest, then the client will successfully authenticate to the server if and only if it has a password $x$ whose hash $H(x)$ is equal to the input of the server $y$. Some of the important features of our protocol are:

- Note that server learns the wire keys corresponding to *one input*. In other words, it is not possible for the server to evaluate circuit $C_i$ on input $x_1$ and circuit $C_j$ ($j \neq i$) on a different input $x_2$. This is rationale behind concatenating the wire keys in step 2 of the protocol.

- Suppose the server denies client's authentication request. Client does not know whether it was denied authentication because it was trying to cheat (by sending invalid circuits) or it had the wrong password $x$.

- Assume that out of the $l$ garbled circuits $C_1, \cdots, C_l$ the circuits with index $j \in B$ (where $B \subseteq \{1, 2, \cdots, l\}$) are not valid. The only way client's cheating is not detected is if the random set $S$ chosen by the server in step 3 is the complement of $B$. Since $S$ is chosen randomly, the probability of this event is $2^{-l}$.

Next we formally argue about the privacy of the client and the server.

**Client's privacy:** Assume that the client is honest and the server is controlled by an adversary $\mathcal{A}$. $\mathcal{A}$'s view consists of the $l$ garbled circuits $C_1, \cdots, C_l$, messages received during the $m$ $OT_1^2$ protocols, all keys for $C_j$ ($j \in S$), where $S \subseteq \{1, 2, \cdots, l\}$ is chosen by $\mathcal{A}$), and keys corresponding to the client's input $x$ for circuits $C_j$ ($j \notin S$). Assume that views corresponding to the $m$ $OT_1^2$ protocols only reveal the secrets corresponding to the input $y$ of $\mathcal{A}$ (let the $\mathcal{A}$ input be $y = y_1 \cdots y_m$ then the server learns $k_{j,k}^{y_k}$ $1 \leq j \leq l$ and $1 \leq k \leq m$). This follows from the privacy of the underlying oblivious-transfer protocol. For example, if one uses the Naor-Pinkas oblivious-transfer protocol, then we have the information-theoretic security for the server. Assuming that the encryption scheme used to construct the garbled circuits is semantically secure, revealing the wire keys for circuits $C_j$ ($j \in S$) does not reveal any information about the client's input. Consider the circuits $C_j$ ($j \notin S$). Server can evaluate this circuit on $(x, y)$ but learns nothing else. Consider an ensemble of garbled circuits $C'_j$ ($j \notin S$), where $C'_j$ computes the constant function $\delta(H(x), y)$.[1]. If the encryption-scheme is semantically secure, then $\mathcal{A}$ cannot distinguish between circuits $C_j$ and $C'_j$ (for $j \notin S$). Essentially $\mathcal{A}$ only learns whether hash of client's password is equal to its input and nothing else.

**Server's privacy:** Assume that the server is honest and the client is malicious. The client is controlled by an adversary $\mathcal{A}$. We construct a simulator Sim which works in the ideal model. Sim acts as the server for $\mathcal{A}$.

- $\mathcal{A}$ sends $l$ copies of the garbled circuits $C_1, \cdots, C_l$ to Sim.
- Sim acts as TP for $\mathcal{A}$ for the $m$ oblivious transfer protocols. Sim knows the inputs of $\mathcal{A}$ (which in the case of the honest client's are the wire keys corresponding to the server's inputs).
- Sim chooses a random set $S_1 \subseteq \{1, 2, \cdots, l\}$ and sends it to $\mathcal{A}$.
- $\mathcal{A}$ sends all the wire keys corresponding to circuits $C_j$ ($j \in S_1$).
- Sim rewinds $\mathcal{A}$, and sends the complement of $S_1$ to $\mathcal{A}$. $\mathcal{A}$ sends all the wire keys corresponding to circuits $C_j$ ($j \notin S_1$). Note that after this step Sim knows the wire keys corresponding to all the garbled circuits $C_1, \cdots, C_l$.
- Sim rewinds $\mathcal{A}$, picks a random set $S \subseteq \{1, 2, \cdots, l\}$, and sends it to $\mathcal{A}$. $\mathcal{A}$ sends wire keys corresponding to the circuits $C_j$ ($j \in S$).
- $\mathcal{A}$ provides the wire keys for all circuits $C_j$ ($j \notin S$). Note that since Sim knows wire keys for all the garbled circuits, it can now construct $\mathcal{A}$'s input $x$.

[1] We assume $C'_j$ is constructed from the same encryption scheme that was used to construct $C_1, \cdots, C_j$.

- Sim checks the validity of all the circuits. If any of the circuits is found to be invalid, Sim sends $0$ to $\mathcal{A}$. Otherwise $x$ is send to the TP and output of the TP is sent to $\mathcal{A}$. Sim outputs whatever $\mathcal{A}$ does.

Let $E_1$ be the event that a honest server detects cheating in the real model, and $E_2$ be the event that Sim detects cheating in the ideal model. It is easy to see that if $E_1$ and $E_2$ are true, then the view of $\mathcal{A}$ in the real model is indistinguishable from the view of Sim in the ideal model. The probability of event $E_1 \wedge E_2$ not happening is bounded by $2^{-l+1}$. Hence we have security for the server in the covert model with $\epsilon = 2^{-l+1}$.

**Replay attacks:**